UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

GRADO EN INGENIERÍA TELEMÁTICA

PROYECTO FIN DE CARRERA

Despliegue de una Infraestructura de Red Definida por Software

Autor: Alberto Gutiérrez Pozuelo
Director: Jaime José García Reinoso

Junio de 2016

Agradecimientos

Nunca es mala ocasión para dar las gracias sin importar el motivo. Por eso en estas líneas quisiera aprovechar la oportunidad que se me brinda para agradecer a todas y cada una de las personas que han permanecido a mi lado y me han dado fuerzas en todo momento a lo largo de esta larga y dura andadura.

A mis amigos, por sacarme siempre una sonrisa al final del día en los momentos difíciles.

A mis profesores, por poner a mi disposición los conocimientos necesarios para formarme académicamente, estando siempre a mi entera disposición. En especial a mi tutor, por brindarme la oportunidad de conocer el mundo SDN mostrándome siempre su apoyo, ayuda y tiempo.

A la universidad, por permitirme formarme no sólo como estudiante sino también como persona, poniendo siempre en mi mano las mejores herramientas para mi éxito.

A todas y cada una de las personas que me acompañaron en la aventura Erasmus, y sin los que mis años como universitario, sin duda, no serían lo mismo.

A mis compañeros de batalla, con los que tantas y tantas veces he pasado horas interminables para sacar una práctica adelante o comprender el temario de un examen.

Pero sobre todo a mi familia, que en una sociedad española en la que el acceso a una educación de calidad es cada vez más difícil, han luchado siempre por darme la oportunidad de labrarme un futuro dándome su apoyo incondicional en las victorias y en la derrotas.

A TODOS ELLOS, GRACIAS.

Para terminar, me gustaría acabar con dos citas simbólicas que reflejan lo que ha sido mi paso por la universidad:

“La cometa se eleva más alto en contra del viento, no a su favor” - Winston Churchill

“El valor de una educación universitaria no es el aprendizaje de muchos datos, sino el entrenamiento de la mente para pensar” - Albert Einstein

Resumen

Desde los inicios de la informática, todo tipo de recursos (computacionales, de almacenamiento o de red) han sido guardados en medios físicos, e intencionadamente separados unos de otros. No fue hasta que se introdujo el concepto (y la demanda) de capacidad económica de computación, almacenamiento y conectividad en el entorno de los data centers, que las organizaciones tuvieron que aunar esfuerzos para agrupar estos recursos. Así surgen los primeros hipervisores o monitores de máquina virtual. Este tipo de tecnología permite que un equipo utilizando un sistema operativo fuera capaz de ejecutar uno o más clientes con los mismos o diferentes sistemas operativos, ahorrando así en costes.

Con la llegada de Internet a cada vez más número de hogares, la demanda aumentó de manera exponencial propiciando que los departamentos IT de cada empresa hayan tenido que ingeniárselas para satisfacer sus necesidades. No obstante, empresas como Amazon, cuyo crecimiento seguía una distribución exponencial (consultar Figura 1), veían cómo cada 6-9 meses se doblaban el número de recursos que necesitaban. De manera que la estrategia que seguían dichas empresas, era comprar mucho más de lo que necesitaban y alquilar los recursos temporalmente a otras empresas dando lugar a los primeros data centers multiusuario, con los consiguientes problemas a la hora de administrar la comunicación entre máquinas para que sólo se pudieran comunicar entre equipos de la misma empresa.

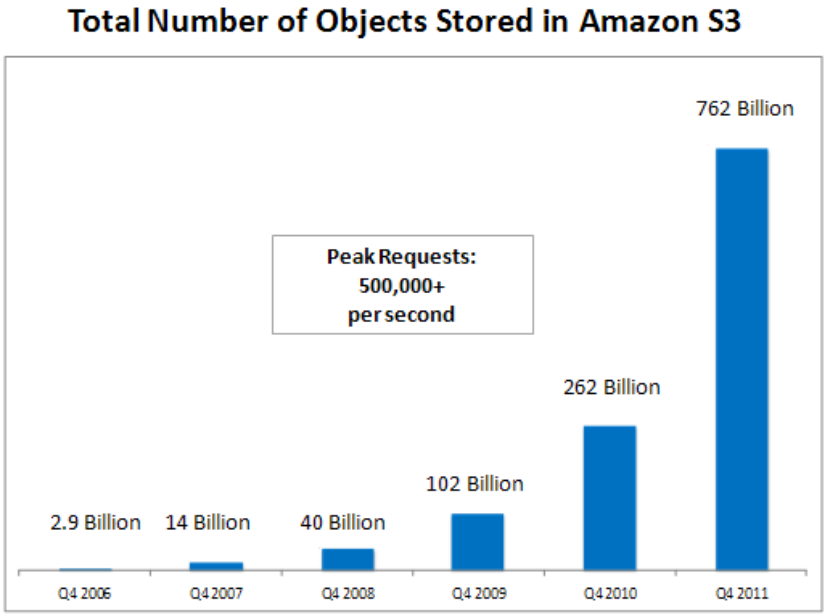


Figura 1: Cantidad de Objetos almacenados en los servidores de Amazon

(Fuente: http://www.statista.com/)

Sin embargo, pese a la evolución de los servicios virtualizados, los equipos de red parecían quedar estancados y sus esfuerzos se dirigían a innovaciones relacionadas con el incremento de la velocidad y la mejora de los protocolos de transmisión de datos, quedando anclados en el uso de IP y MPLS como protocolos de comunicación.

En consecuencia, se empieza a replantear el enfoque dando lugar a nuevas ideas sobre las que se sustentan los cimientos, de lo que a la postre se conocería como redes definidas por software (SDN del inglés Software-Defined Networking). Las SDN surgen como respuesta a una falta de innovación y evolución en los dispositivos de red por parte de los proveedores (ya que se necesitaba de una gran maña y dedicación para poder probar nuevos protocolos o configuraciones en routers comercializados) unido a una significante reducción de costes.

Las SDN aportan un nuevo enfoque a la administración de la red, ya que los planos de control (parte lógica o cerebro de la red) y de datos (plano encargado del reenvío de paquetes) son separados dando lugar a un modelo centralizado. Otro de los conceptos clave, es que ambos planos son directamente programables, y mediante el uso de APIs los administradores de red pueden adecuarlas a sus necesidades fácilmente. Como resultado, tenemos una arquitectura extremadamente dinámica, fácilmente administrable, adaptable y de coste ajustado.

Como podemos ver en la Figura 2, una arquitectura SDN se compone por lo tanto de dos planos principales que abarcan tres capas. Por un lado tenemos la capa de infraestructura (infraestructure layer), que se corresponde con el plano de datos. En ella, se encuentran todos los dispositivos de red (switches y routers) encargados de reenviar flujos de datos, en base a unas tablas cuyas entradas son configuradas por el plano de control. Por encima de esta capa, nos encontramos la capa de control (control layer). Y es aquí, dónde se encuentra lo que se conoce como controlador SDN. Este, dispone de una vista abstracta de la red, y tiene como principal función la de cumplir con las políticas de red establecidas manualmente por un administrador o de manera dinámica por las aplicaciones SDN. Para ello, se comunica con los switches que tiene por debajo y añade entradas a sus tablas de flujo para indicarles cómo deben manejar los paquetes que lleguen. Por último, tenemos la capa de aplicación (application layer) y que forma parte también del plano de control. En ella se encuentran las aplicaciones SDN, las cuales se comunican con el controlador para transmitir las políticas de red a seguir, en función de las necesidades de cada empresa.

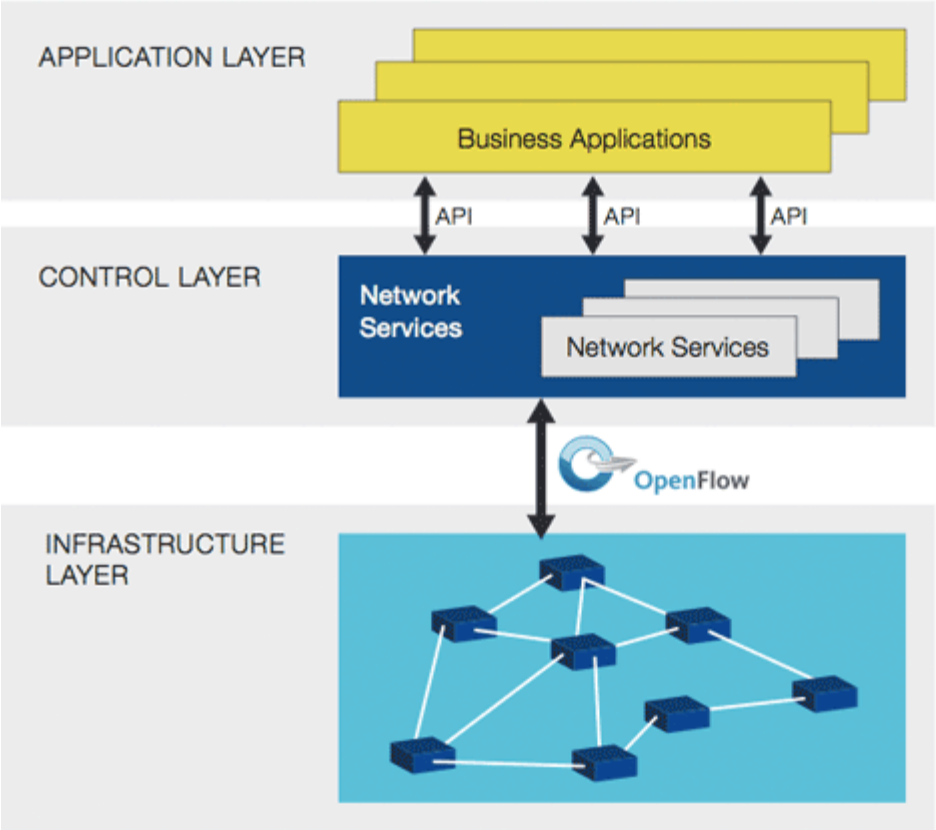


Figura 2 : Arquitectura Básica SDN

(Fuente: http://www.opennetworking.org)

La comunicación entre estas capas se lleva a cabo mediante el uso de APIs Northbound (o "hacia el norte") y Southbound (o "hacía el sur").

Las Northbound APIs se utilizan para la comunicación entre el controlador y las aplicaciones SDN. Mediante esta interfaz, las diferentes aplicaciones comunican al controlador las políticas a implementar en la red de manera automatizada. También, se utiliza para que el controlador informe de posibles cambios en la arquitectura de red, y así solicitar nuevas instrucciones a ejecutar en función de cada situación. Son muchas las interfaces utilizadas, las cuales varían según el controlador y las aplicaciones, provocando que a veces se genere cierta incompatibilidad.

Las SouthBound APIs facilitan al controlador la comunicación con los equipos de red. Mediante el uso de estas interfaces, el controlador puede realizar configuraciones dinámicas añadiendo, modificando o eliminando entradas en la tabla de flujo de cada dispositivo para satisfacer las demandas y necesidades en tiempo de ejecución. La API más extendida en este ámbito es OpenFlow [1] , un protocolo de estándares abiertos que surgió como fruto de la invesitgación de un grupo de ingenieros de la Universidad de Stanford.

En 2011, un conjunto de los proveedores de servicios más influyentes creó la ONF (Open Networking Foundation) [2]. Organización cuyo propósito es la estandarización,

comercialización y promoción del uso de OpenFlow en redes de producción. Una de las ventajas del uso del protocolo OpenFlow, es que una red puede ser gestionada como un único objeto, permitiendo implementar configuraciones o actualizaciones de manera programada y automatizada.

Dentro del mercado actual, podemos encontrar tanto switches que únicamente utilizan el protocolo OpenFlow, como híbridos que además son capaces de entender y asimilar los protocolos de enrutamiento convencionales (RIP, OSPF,…). Este tipo de switches, suelen utilizarse en las fronteras entre una arquitectura SDN y un esquema de red tradicional. En la figura 3, podemos observar un breve diagrama del funcionamiento de un switch OpenFlow, cada vez que recibe un paquete en cualquiera de sus puertos.

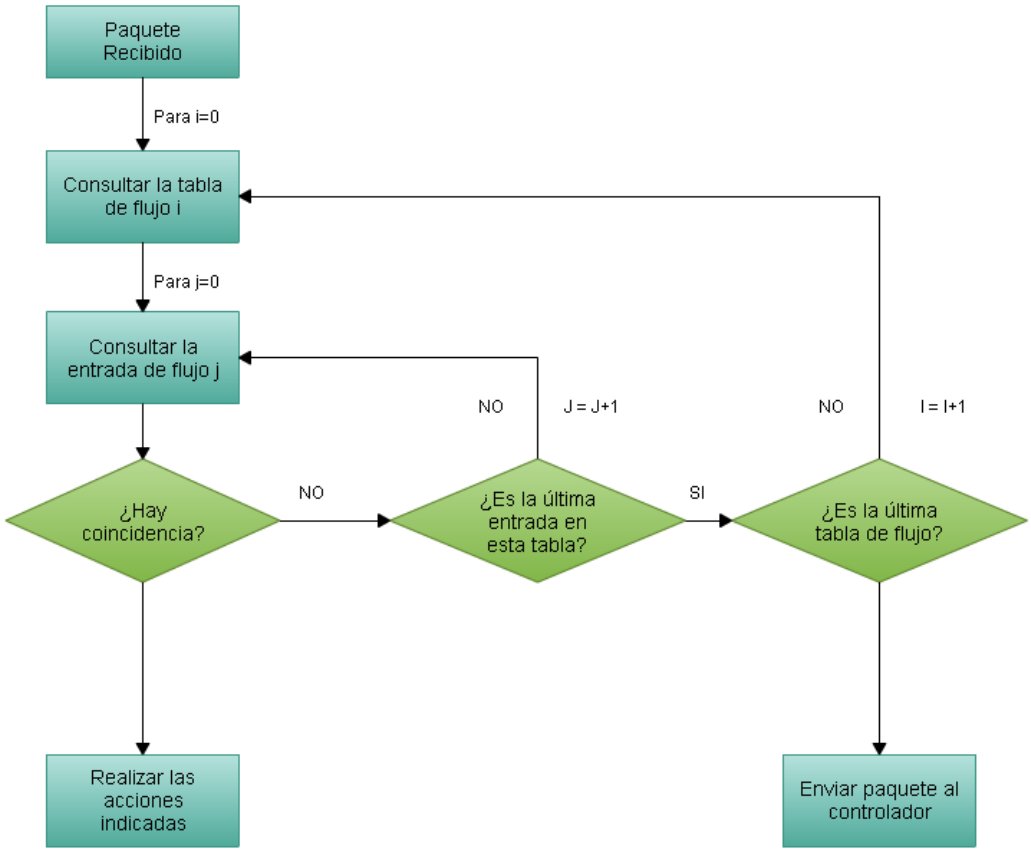


Figura 3 : Diagrama de Flujo de un Switch OpenFlow

(Fuente: Elaborada para este TFG)

Entendemos por flujo, todo grupo de paquetes que comparten una serie de requisitos, y que por lo tanto pueden tener una concordancia en la tabla de flujo. A modo de ejemplo, si nuestra tabla de flujo tiene una entrada cuya condición es que los paquetes que se reciban por el puerto 1 se descarten, todos estos serán descartados sin importar su IP origen/destino, MAC u otros campos. Sin embargo, supongamos que existe otra entrada

de flujo con una prioridad mayor a la anterior, y que indica que los paquetes con IP destino 10.0.0.1 no se descarten, sino que se reenvíen por el puerto 2. En este caso prevalecería la entrada con mayor prioridad, y los paquetes con destino la IP indicada se reenviarían correctamente (el resto se seguirían descartando).

Para comprobar el funcionamiento de todo lo expuesto previamente, se pueden utilizar diferentes métodos: desde el uso de herramientas como Mininet (software que permite disponer de un entorno virtualizado de red), pasando por el uso de equipos físicos o (como es el caso de este proyecto) utilizando máquinas virtuales configuradas adecuadamente para ello.

A lo largo de este proyecto se ha elaborado un guion de prácticas, que ha sido dividido en 4 partes o laboratorios principales. Nuestro objetivo, es que un alumno con unos conocimientos básicos de lo que es una arquitectura SDN, pueda desplegar un escenario sencillo con el que hacer pruebas y asimilar los conceptos más relevantes de las redes definidas por software.

La implementación de los escenarios utilizados, se llevará a cabo mediante el uso de máquinas virtuales con una distribución Linux. Para ejecutarlas, utilizaremos VirtualBox[3] (software de virtualización de Oracle). Entre las máquinas utilizadas podemos diferenciar tres tipos principales:

- **Equipos:** Estas máquinas virtuales se utilizarán en los diferentes escenarios como clientes/servidores de una arquitectura.

- **Switch OpenFlow:** Para simular el funcionamiento de un switch OpenFlow, se dispone de este tipo de máquina en la cual se ha instalado Open Virtual Switch. OVS es un software de código abierto diseñado para ejercer las labores de un switch virtualizado y compatible con el protocolo OpenFlow.

- **Controlador OpenDaylight:** Para nuestra máquina virtual que desempeñará la función de controlador SDN, disponemos de diferentes distribuciones de OpenDaylight [4] (tanto a nivel de usuario como a nivel de desarrollador). ODL es una plataforma de código abierto que cuenta con el apoyo de la Linux Foundation [5], y que es uno de los controladores más utilizados en las arquitecturas SDN debido, entre otros, a la gran comunidad de usuarios que tiene detrás.

Dentro de las pruebas llevadas a cabo, para posteriormente redactar el guion de prácticas final, se pueden diferenciar cuatro partes:

Laboratorio 1:

En esta parte, dispondremos de un escenario compuesto por 3 equipos que actúan indistintamente como clientes o servidores, y un switch al que están

conectados. A pesar de no disponer de un controlador como tal, seremos nosotros mismos los que simulemos sus funciones añadiendo entradas a la tabla de flujos del switch manualmente.

El objetivo principal de esta práctica, es entender cómo afectan las entradas de flujo a la lógica de reenvío de un switch OpenFlow. Podremos analizar los flujos y sus diferentes campos, así como tener una pequeña toma de contacto con el software de OpenVSwitch. Para, finalmente, simular el funcionamiento de un firewall configurando las entradas necesarias.

Laboratorio 2 :

La arquitectura utilizada en esta práctica será la misma que en el laboratorio anterior, pero esta vez añadiendo un controlador OpenDaylight.

Esta parte, tiene como objetivo familiarizarnos con el entorno de OpenDaylight a través del uso de la interfaz de línea de comandos (Karaf) y de su interfaz gráfica (Dlux) (disponible a través de la dirección http://<IP-controlador>:8181/index.html una vez arrancado el controlador).

Analizaremos también los diferentes tipos de configuraciones y programación de flujos, utilizando herramientas que hacen uso de la API de REST (Transferencia de Estado Representacional) .

Laboratorio 3 :

En este tercer escenario, simplemente añadiremos a la arquitectura anterior un host que actuará como cliente, ya que el resto lo harán como servidores.

Añadiremos una lógica al controlador vía REST API, que simulará el funcionamiento de un balanceador de carga sencillo. Este seguirá un algoritmo de Round-Robin para enrutar las peticiones TCP a los diferentes servidores.

Laboratorio 4 :

Por último, utilizando una arquitectura similar a la del Laboratorio 2 (utilizando esta vez la versión para desarrolladores de OpenDaylight), aprenderemos cómo instalar una aplicación externa que simulará las funciones de un router de capa 2. Dado que no estaba entre los objetivos del proyecto, se utilizará una aplicación creada por el grupo de desarrolladores de SdnHub.

Palabras Clave:

SDN, OpenFlow, switch, OpenDaylight

Abstract

Up until a few years ago, storage, computing, and network resources were intentionally stored in hardware, remaining separated one from each other. It was really only after the introduction (and demand) of low-priced computing power, storage and networking in data center environments that organizations were forced to cooperate in order to group them all together. Thereby, first hypervisors appeared. This kind of technology allowed a host running a specific operating system to execute one or more client operating systems saving on costs.

With the expanding of the Internet, the demand increased exponentially. Consequently, IT departments contrived to satisfy their lacks. Nevertheless, companies like Amazon were growing at the rate of an exponential distribution doubling every six to nine months (see Figure 1). As a result, Amazon's IT department was forced to acquire large quantities of resources that they would temporally rent to other companies until they needed. So was the birth of multitenant datacenters. This of course created a new problem, which was how to separate thousands of potential tenants, whose resources needed to be spread across different physical machines.

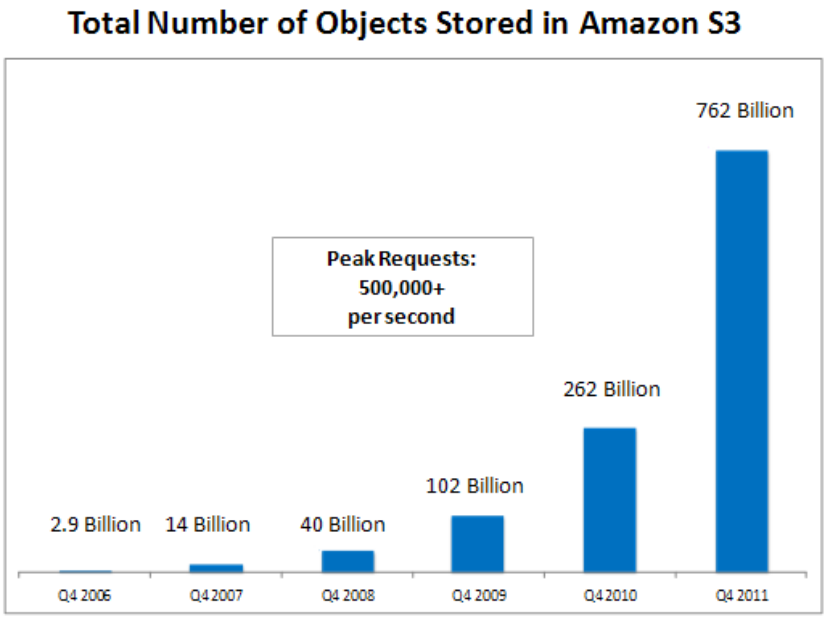


Figure 4 (English Version): Total Number of Objects Stored in Amazon Servers

(Source: http://www.statista.com/)

In contrast to virtualized services evolution, network equipment seemed to get stuck in terms of innovations beyond the increase of feeds and speeds. Communication protocols had not changed much since the advent of IP and MPLS.

Therefore, new paradigms are brought to scene giving place to the core of Software-Defined Networking (SDN) concepts. SDN emerge as a response to a lack of innovations in network devices, in which vendors were not meeting the needs in programmability and configuration tools of the organizations.

SDN solutions separate control plane and data plane, providing a new approach to network management and making it easy to program them using available APIs. Consequently, we have dynamic, economic and configurable network architecture.

As we can see in Figure 2, SDN architecture is composed of two main planes including three layers. First of all, at the bottom, we have the infrastructure layer in which we can include all forwarding devices - switches and routers –. Above this one, we find the control layer, responsible for accomplishing network policies established by the applications and where the SDN Controller resides. To do so, it communicates with network elements in order to create new flow entries to handle incoming packets. Finally at the top, we dispose of the application layer, whose main purpose is to pass down information about network policies to be implemented to the controller having into account each company's needs. The way these three layers communicate among them is through Northbound and Southbound APIs.

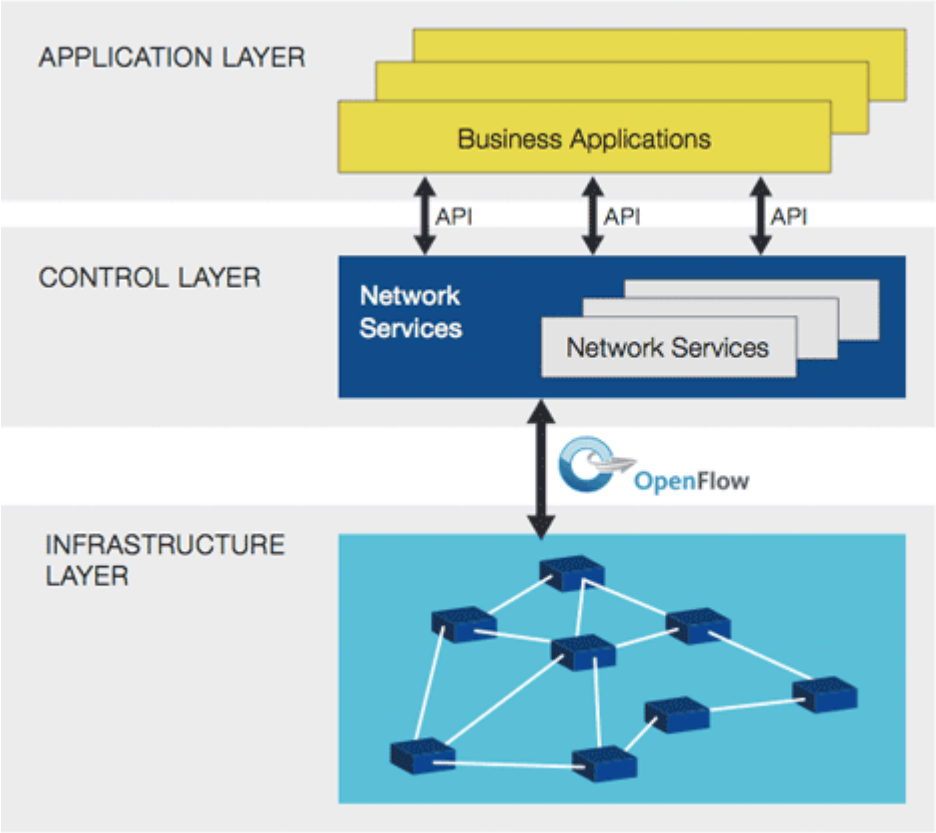


Figure 5 (English version) : Basic SDN Architecture

(Source: http://www.opennetworking.org)

On the one hand, Northbound APIs establish communication between SDN applications and the controller. By means of this interface, applications are able to directly express to the controller the desired network behavior and requirements. Moreover, it is also used to request modifications on topology changes. Many different interfaces are available in the market, what leads into an incompatibility issue.

On the other hand, Southbound APIs make communication among network devices and controller much easier. By means of this interface, the controller dynamically configures flow tables adding, modifying or erasing entries in every single switch. The most widespread API in this context is Open Flow [1], which is a standardized open protocol created by a group of network researchers from Stanford University.

In 2011, a nonprofit consortium called the Open Networking Foundation (ONF) [2] was formed by a group of service providers to standardize, commercialize and promote the use of Open Flow. What makes Open Flow so interesting is the fact that network administrators have the tools to automatically program updates and configurations.

In today's market it is possible to find different Open Flow-related devices. From hybrid switches - capable of Open Flow understanding as well as conventional use of routing protocols - to Open Flow-only switches. In figure 3, we can observe the path covered by an ingress packet in an Open Flow switch.

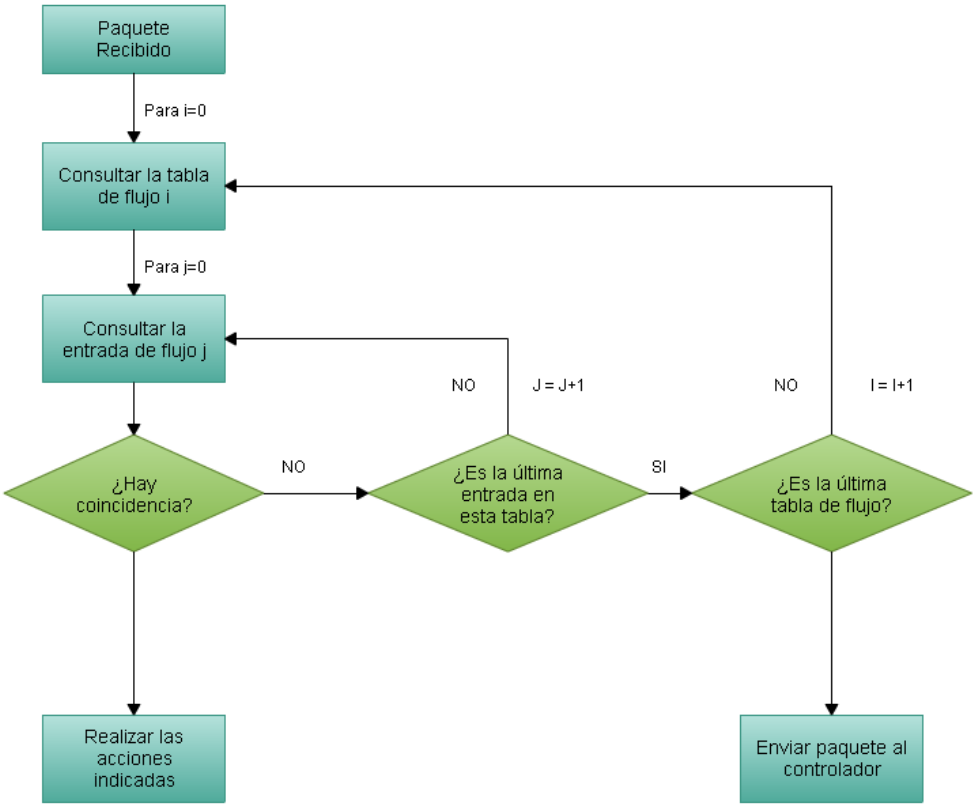


Figure 6 (English Version) : Flowchart in an OpenFlow Switch

(Fuente: Elaborada para este TFG)

When we talk about a traffic flow, we are referring to any group of packets sharing a set of requirements, and therefore may have the same match in the flow table. As an example: If we have a flow table with an entry, which says all packet received in port 1 must be dropped, every incoming packet at this port will be discarded no matter neither its MAC nor its IP was. However, if there is another flow entry with a highest priority that commands to forward packets with IP destination address 10.0.0.1 to port 2, this flow traffic won't be dropped.

In order to probe SDN environments, several methods are accessible: we can use Mininet – network virtualizing software - , physic equipment or – as we do in this project – by means of virtual machines.

During this project, a practice manual was developed and divided into four labs. It has been thought to be easy to perform by a student with basic knowledge on SDN concepts, being able to deploy a simple Software-Defined Networking architecture. For its implementation, three different types of virtual machines -running a Linux distribution under Virtual Box [3] - are ready to use:

- **Hosts:** This virtual machine is intended to be used as a client/server in an SDN topology.

- **Open Flow Switch:** This virtual machine is intended to be used as an Open Flow switch simulating its behavior by means of the Open Virtual Switch software. OVS is an open source project designed to virtualize a switch.

- **OpenDaylight Controller:** This virtual machine is intended to be used as an SDN controller. Two different VMs will be available. Both will have an OpenDaylight distribution installed with two different versions –user and developer-. ODL is an open source platform supported by The Linux Foundation [5] and with a large community behind them.

Among the tests taking place in order to write our manual, we can differentiate four labs:

Lab 1:

In this part we will have a scenario formed by 3 hosts attached to one switch. Despite the fact of not having a specific controller, we will perform that task ourselves by adding new flow entries to the switch.

The main objective of this practice is to understand how flow entries work. Entries will be added, modified and properly analyzed. Moreover, it will also be possible to deal with some OpenVSwitch configuration. Finally we will gather all knowledges acquired in order to simulate a TCP Firewall.

Lab 2:

The architecture in this part will have the same structure as the one used in last laboratory, but adding an OpenDaylight Controller.

Its main goal is to get in touch with OpenDaylight CLI –Karaf- and GUI –Dlux; available in http://<controller-IP>:8181/index.html-.

REST APIs will be also covered in this chapter and we will learn how to use them in order to configure flows.

Lab 3:

In this third scenario we will simply add a new host to the previous topology. This host will work as a client meanwhile the others will do it as a server.

By using a REST API we will configure a simple load balancer following a round-robin algorithm.

Lab 4 :

Finally, and using the topology of lab 2 we will learn how to install a Java application developed by SdnHub Group. This app will add to the controller the logic of a label 2 switch.

Key Words:

SDN, OpenFlow, switch, OpenDaylight

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

Este apartado proporciona una breve introducción a las Redes Definidas por Software (SDN de sus siglas en inglés Software-Defined Networking), y pretende explicar, por un lado por qué las necesitamos y por otro mostrar por qué han generado un gran entusiasmo en el sector de las telecomunicaciones.

Adicionalmente, expondremos los motivos y objetivos que nos han llevado a realizar este proyecto.

1.1 INTRODUCCIÓN

Desde hace años internet forma parte de nuestras vidas, y poco a poco se ha ido introduciendo en diferentes ámbitos, hasta llegar a estar presente en casi todas partes. En consecuencia, las redes se han convertido en un elemento absolutamente esencial en el mundo moderno.
Actualmente, han ido evolucionando y podemos diferenciar varios modelos: desde redes completamente "on premise" (redes convencionales diseñadas previamente a su implantación, y cuya modificación requiere un cambio a nivel físico), pasando por las basadas en la computación en la nube e incluso híbridos de ambas. Estas, proporcionan los enlaces básicos de comunicación para que las organizaciones puedan ejecutar sus aplicaciones, entregar servicios y ser competitivas. Las redes definidas por software (SDN) representan un nuevo paradigma en cuanto a la forma en que las redes operan, son controladas o configuradas.

Mucha gente difiere acerca de la correcta definición de SDN. Esto es debido a que a medida que la tecnología ha ido evolucionando, se han introducido nuevos aspectos y soluciones que redefinen el concepto principal. Por lo general, la acepción más clásica contempla las redes definidas por software como redes dirigidas por aplicaciones y controladores SDN, sustituyendo a las tradicionales consolas de gestión de red y sus comandos, los cuales podían llegar a ser tediosos cuando se trataba de administrar un sistema a gran escala.

Con la introducción de este revolucionario concepto, se comprobó cómo tareas complejas a nivel IT, que anteriormente implicaban un duro esfuerzo y dedicación, podían ser ahora automatizadas con el uso de software desarrollado siguiendo las bases de SDN de manera más sencilla y eficaz.

Cuando se trajo a escena por primera vez la idea, se planteó como una serie de conceptos más rígidos, que contemplaban cómo debía de estar estructurada y diseñada una arquitectura SDN y qué se consideraba una solución del tipo SDN. Sin embargo,

hoy en día cada cliente toma una vista mucho más amplia sobre qué tipo de solución se adecua mejor a sus necesidades.

Otra de las grandes ventajas es que se trata de una tecnología libre (que no requieren de autorización o licencia para su uso). Esto permite una mayor innovación, flexibilidad , interoperabilidad y soluciones más rentables. Si una red está regida bajo los estándares apropiados, puede ser manejada por varias aplicaciones SDN ejecutándose en un solo controlador. Este modus-operandi es mejor que el actual, en el cual cada dispositivo de red administra su propia consola y comandos aumentando la complejidad, y donde las opciones de configuración vienen delimitadas por el distribuidor.

En conjunto con la evolución de las redes definidas por software, existen un gran número de tendencias tecnológicas que afectan al diseño y arquitectura de los data center modernos. En la mayoría de las organizaciones, los data center están migrando de un modelo tradicional de cliente-servidor, a uno en el que un mayor volumen de datos es transferido dentro del data center (frecuentemente conocido como east-traffic). Esto requiere de una mayor escalabilidad de red y políticas más sofisticadas para la asignación de recursos. A todo esto, se suma que una gran mayoría de departamentos IT están mostrando gran predisposición por pasar a un entorno (ya sea público, privado o híbrido) basado en la nube.

Los servicios públicos en la nube ofrecida por compañías como Amazon, Microsoft o Google, han permitido a los departamentos corporativos IT vislumbrar un método de autoservicio, a la par que demostrar lo dinámicas que estas aplicaciones y servicios pueden ser. De ahí que el resto de organizaciones ahora exijan los mismos niveles a sus propios departamentos IT. De hecho, se considera que las SDN son un factor clave para conseguir aumentar la agilidad de las tecnologías IT, así como mejorar su capacidad de autoservicio.

Las empresas, además, están apostando fuertemente por las aplicaciones del tipo Big Data, las cuales requieren de una gran capacidad de procesado en paralelo, junto con cientos o miles de servidores. Todas estas peticiones, y sus respectivas respuestas, traen consigo una consecuencia: la congestión de la red. La demanda de una tecnología capaz de lidiar con este tipo de tráfico de manera eficiente, ágil y con un alto nivel de rendimiento ha desembocado en el uso de soluciones SDN.

Los beneficios a un más alto nivel de una estrategia SDN repercuten en todas las áreas de una organización. Se obtiene una ventaja competitiva, ya que la infraestructura es más productiva, aumenta la velocidad del negocio, reduce el coste total de propiedad y minimiza los riesgos gracias a una mejor seguridad.

Para tener una mejor perspectiva de por qué las SDN han tomado tal importancia, es necesario conocer el escenario anterior. Las arquitecturas de red tradicionales tienen una serie de limitaciones significativas que han de ser superadas para satisfacer las

necesidades IT modernas. Las redes actuales deben ampliarse para poder soportar grandes cargas de trabajo con mayor agilidad y, al mismo tiempo, mantener bajos los costos. Pero el enfoque tradicional tiene importantes limitaciones:

- **Complejidad:** La abundancia de protocolos y funcionalidades de red para casos de uso específicos, propicia el aumento de la complejidad de la red. Las tecnologías antiguas solían reciclarse con rápidos parches para adaptarlas a los requisitos del negocio. Las funciones de los dispositivos de red quedaban encorsetadas bajo la exclusividad de cada proveedor o eran implementadas mediante comandos patentados.

- **Políticas incoherentes:** Las políticas de seguridad y calidad de servicio (QoS) de las redes actuales, deben configurarse manualmente o ejecutando scripts en cientos o miles de dispositivos de red para actualizarlos. Esto ocasiona que los posibles cambios, sean extremadamente complicados de implementar para una organización sin que ello implique una inversión considerable en el desarrollo de scripts, o en herramientas que puedan automatizar el proceso de cambios en la configuración. Las configuraciones manuales son susceptibles de errores, y pueden llevar a horas y horas de diagnóstico para encontrar la línea de la política de seguridad o lista de control de acceso (ACL) que se introdujo de manera incorrecta, en un dispositivo determinado. Además, cuando las aplicaciones son eliminadas es prácticamente imposible borrar todas las políticas asociadas a todos los dispositivos, lo que aumenta aún más la complejidad.

- **Incapacidad de expansión:** A medida que las cargas de trabajo de las aplicaciones cambian, la demanda de ancho de banda en la red aumenta. En consecuencia, los departamentos IT necesitan una red estática con exceso de suscripciones o bien crecer con las exigencias de la organización. Lamentablemente, la mayoría de las redes tradicionales se aprovisionan en modo estático. De manera que, aumentando el número de terminales, servicios o ancho de banda, es necesario planificar y rediseñar de forma sustancial la red.

Tras un breve posicionamiento sobre el escenario actual, podemos ver que se presentaban una serie de inconvenientes que, aunque quizás podían no ser críticos, si eran muy molestos. Y precisamente para rellenar ese hueco nacieron las SDN.

En un principio, hubo mucha publicidad en torno a las SDN antes de comprender los casos de uso reales. Pero lo que realmente catapultó la evolución, fue la certeza de que muy pocas organizaciones tenían realmente la capacidad, el deseo y los incentivos necesarios para elaborar una nueva clase de aplicaciones para programar la red. La gran mayoría de organizaciones simplemente buscaban automatizar las tareas IT, acelerar la implementación de aplicaciones, aportar más flexibilidad a las redes de la nube y ajustar mejor su infraestructura IT a los requisitos empresariales. No obstante, el enfoque ha cambiado; las SDN son plataformas capaces de almacenar un sinfín de soluciones para

automatizar y organizar el flujo de trabajos IT. Los administradores de red no necesitan desarrollar nuevas aplicaciones, simplemente disponen de un mercado dónde adquirirlas ya implementadas, y listas para ser ejecutadas bajo tecnologías SDN. [6]

Los principales rasgos de una arquitectura SDN son:

- **Programables**: Gracias a la separación del panel de datos y de control, la red se hace fácilmente programable.

- **Ágil:** Los administradores de red pueden ajustar el ancho de banda de forma dinámica, para satisfacer posibles demandas de servicio.

- **Centralizada:** Se concentra la inteligencia de la red en un único elemento (controlador), el cual tiene una visión completa de la red, haciendo que para las aplicaciones sea como un único switch.

- **Configuración programable:** SDN permite a los operadores de red configurar, administrar, optimizar y asegurar los elementos de la red rápidamente usando programas automatizados que pueden ser desarrollados por ellos mismos.

- **Basado en estándares abiertos:** Al usar un controlador SDN con licencia libre se facilita su implementación, ya que las instrucciones son facilitadas por un único agente en lugar de múltiples proveedores.[7]

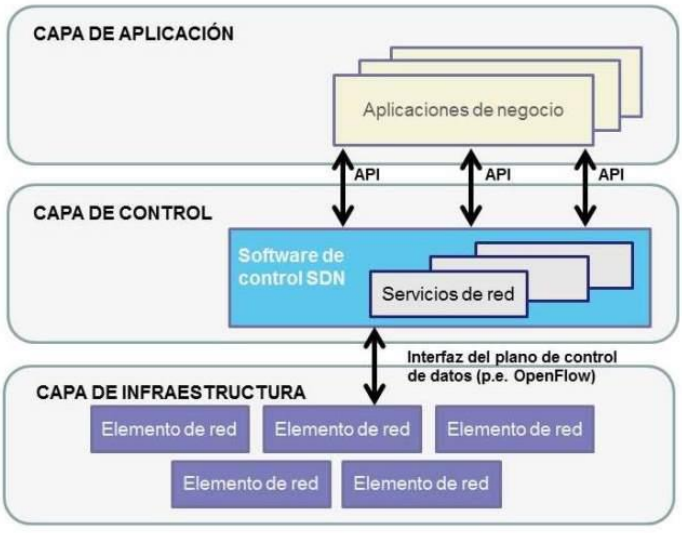


Figura 7: Arquitectura SDN (2)

(fuente: http://www.ramonmillan.com/tutoriales/sdnredesinteligentes.php)

1.2 MOTIVACIÓN Y OBJETIVOS

Este proyecto tiene como motivación principal, el ahondar en el funcionamiento de las redes definidas por software. Para ello, se desplegará una infraestructura de red SDN sencilla llevando a cabo su diseño e implementación.

*El **objetivo principal** del proyecto es llevar a cabo una serie de pruebas y ejercicios básicos, que a la postre darán lugar a un guion de prácticas que podrá ser utilizado por un alumno de máster para asimilar los conceptos más relevantes de las redes definidas por software.*

Para la implementación de los escenarios utilizados, se han usado máquinas virtuales configuradas específicamente para ello con una distribución Linux. Al alumno se le facilitarán varios tipos máquinas virtuales para desplegar el escenario necesario en cada laboratorio. Podemos diferenciar los siguientes tipos:

- **PC_XX:** Estas máquinas virtuales se utilizarán en los diferentes escenarios como clientes/servidores de una arquitectura y deberán ser configuradas sus interfaces adecuadamente como se indique en el manual.

- **OVS_XX:** Para simular el funcionamiento de un switch OpenFlow, se dispone de este tipo de máquina en la cual se ha instalado Open Virtual Switch. OVS es un software de código abierto diseñado para ejercer las labores de un switch virtualizado y compatible con el protocolo OpenFlow. Será necesaría la configuración del bridge utilizando los comandos aportados

- **Basic Controller:** Para nuestra máquina virtual que desempeñará la función de controlador SDN, disponemos de diferentes distribuciones de OpenDaylight (tanto a nivel de usuario como a nivel de desarrollador). ODL es una plataforma de código abierto que cuenta con el apoyo de la Linux Foundation, y que es uno de los controladores más utilizados en las arquitecturas SDN debido, entre otros, a la gran comunidad de usuarios que tiene detrás.

Dentro de las pruebas llevadas a cabo, para posteriormente redactar el guión de prácticas final, se pueden diferenciar cuatro partes:

Laboratorio 1: El objetivo principal de esta práctica, es entender cómo afectan las entradas de flujo a la lógica de reenvío de un switch OpenFlow

Laboratorio 2: Esta parte, tiene como objetivo familiarizarnos con el entorno de OpenDaylight a través del uso de la interfaz de linea de comandos (Karaf) y de su interfaz gráfica (Dlux).

Laboratorio 3: Añadiremos una lógica al controlador, que simulará el funcionamiento de un balanceador de carga sencillo.

Laboratorio 4: Aprenderemos cómo instalar una aplicación externa.

1. INTRODUCTION (English Version)

This chapter pretends to be an introduction to Software-Defined Networking (SDN) explaining why are so interesting and why do we need them.

In addition, main objectives and motivations for the performance of this Bachelor Thesis will be exposed.

1.1 INTRODUCTION

Over recent years, the Internet has grown in importance until forming part of our day by day life. Consequently, telecommunications networks have become an absolutely essential element of the equation in today's modern world.

Nowadays, we can distinguish three different models: conventional networks so-called on premise, cloud-based networking and a hybrid of both. All these networks provide basic communications links that organizations can use to run their applications, supply services and be competitive. Software-Defined Networking (SDN) represents a new paradigm in the way networks are configured or controlled.

Many people differ about the proper definition of SDN due to its continuous evolution. Frequently, SDN is conceived as a network controlled by software applications and SDN controllers instead of traditional consoles, whose management usually required a lot of administration on a large scale topology.

With the appearance of this new concept, IT departments had the capability to automate complex tasks making them much easier and more efficient than before using SDN solutions.

When SDN first come into view, there were more rigid ideas about how SDN architectures should be designed and deployed. However, in today's market each customer takes a wider view of what kind of SDN solution better fits their needs.

Moreover, another of the great advantages of SDN is the fact that it is an open technology. This leads to greater innovation, flexibility and cost-effective solutions. As a result, a network operating under proper standards can be managed by several SDN applications running in one controller. Compared with current architectures, in which configuration options are delimited by vendors and every network element has to manage its own console making it way more complex, it is a far better solution.

Along with the evolution to SDN, there are a huge amount of trends affecting the design and architecture of modern data centers. Most organizations are moving from traditional client-server architectures to models in which more data is being transferred within the data center (east-west traffic), leading into the demand of more scalability and more sophisticated network policies for resource allocation. Moreover, many IT departments are showing great interest in shifting to cloud environments.

Public cloud services offered by companies such as Amazon, Microsoft or Google have given IT departments a glimpse of self-service while demonstrating how agile and dynamic these applications can be. That is why organizations are now demanding same levels from their own IT departments. SDN in fact, is being looked at as an essential point in order to increase agility and self-service capacity in IT technologies.

Companies are also investing in Big Data solutions, whose implementation demands lots of parallel processing power and hundreds or thousands of servers. All these requests and responses may lead to congestion in the network. Because of that, organizations are demanding technologies capable of handle this situations in an efficient, agile and high-performance method. SDN solutions are intended to meet those demands.

The top perks of an SDN strategy have an effect on all areas of the organization. Increasing productivity and business speed, and decreasing the total cost of the infrastructure enterprises receive a competition advantage minimizing the risks.

To better understand why SDN has become so important, it is advisable to know the previous scenario. Traditional architectures have several limitations that must be overcome in order to satisfy modern IT requirements. Current networks must be expanded to deal with huge workloads with greater agility, while keeping costs at a low level. Nevertheless, the traditional approach has significant limitations:

- **Complexity:** The abundance of networking protocols and functionalities for the same use cases leads to an increase of complexity in the network. Old technologies were usually recycled adding new elements fixing potential problems. Network features were limited to vendors or were implemented through proprietary commands.

- **Inconsistent policies:** Security and quality-of-service (QoS) policies must be configured manually or scripted across hundreds or thousands of network elements in order to update them. This makes change implementations extremely complicated and expensive.

  Manual configuration is susceptible to human errors with the consequent time spent to fix that failure in a specific device. In addition, when applications were erased it was almost impossible to delete all associated policies from each

device

- **Inability to scale:** As application workloads change, bandwidth demand increases. As a result, IT departments need a static network with lots of subscribers or to grow with the demands of the organization. Unfortunately, the majority of traditional networks are provisioned in such a way that increasing the number of endpoints, services or bandwidth requires planning and redesign of the network.

After a short introduction about today's scenario, we are able to appreciate current network drawbacks, which even not being a critical point made things quite annoying. Because of that, SDN came to stay.

For a long time, SDN received a lot of attention but nor really understanding its potential. What really launched to fame SDN was the certainty that few organizations had the capacity, desire and stimulation to develop a new set of applications that allowed programming the network. They were just looking for IT tasks automation, reduction of application developing time, increase of cloud computing networks flexibility and business requirements adjustments. Therefore, deploying SDN topologies in their organizations, network administrators are able to simply download plug-and-play applications already developed and based on SDN technologies. [6]

Following main attributes of SDN architecture are presented:

- **Programmable:** Due to control and data planes separation the networks can be easily programmable.

- **Agile:** Network administrators can dynamically adjust bandwidth to satisfy possible service demands.

- **Centralized Structure:** Network control intelligence is logically centralized using an SDN controller. The controller has an abstract view of the whole network.

- **Automated Configuration:** SDN allows network administrators to easily and quickly configure, manage, optimize and secure network elements using automated programs.

- **Standard-based:** SDN is based on standard APIs. This allows unification, since all network devices – regardless vendors- use the same commands.[7]

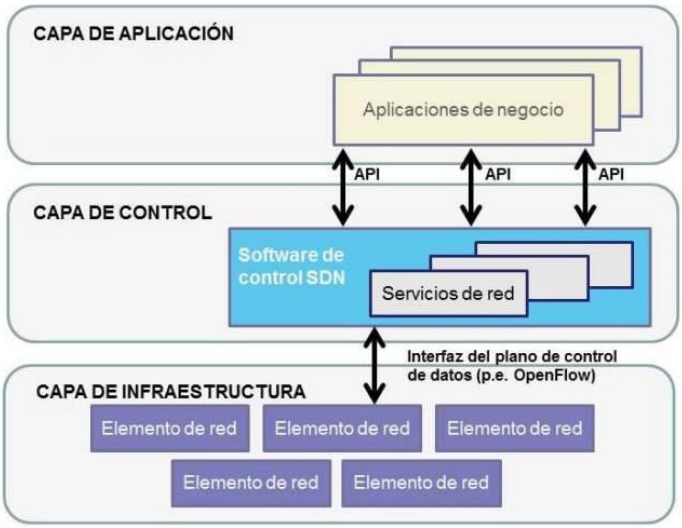


Figure 8 (English Version): SDN Architecture (2)

(Source: http://www.ramonmillan.com/tutoriales/sdnredesinteligentes.php)

1.2 MOTIVATION AND OBJECTIVES

This Bachelor Thesis intends to deep into Software-Defined Networking operation. With this purpose, we will deploy a simple SDN architecture carrying out its design and implementation.

The main goal *is to carry out tests and basic exercises that will eventually result into guide notes available to be used by a master student in order to understand the basics of SDN technologies.*

For implementation specially configured virtual machines running a Linux distribution will be used. Among them we can find different types:

- **PC_XX:** This virtual machine is intended to be used as a client/server in an SDN topology. Its interfaces must be manually configured according to guidelines.

- **OVS_XX :** This virtual machine is intended to be used as an OpenFlow switch where Open Virtual Switch will be installed in order to simulate that behavior. OVS is an open source licensed software able to act as an OpenFlow switch. Bridge configuration following guidelines will be needed.

- **Basic Controller:** These virtual machines are intended to be used as an SDN Controller. Several versions of OpenDaylight controller are available depending on the level required – user or developer- . ODL is an open source platform powered by the Linux Foundation and one of the most SDN controllers used.

Regarding the guide notes provided, we can divide them in four parts:

Lab 1: The main objective of this part is to understand OpenFlow entries and how they affect the behavior of an OpenFlow switch.

Lab 2: This laboratory focuses on get used to OpenDaylight controller using its command line interface –Karaf- as well as its web interface -Dlux- .
Lab 3: At this part we will add simple load balancer logic to the controller.

Lab 4: Finally we will learn how to install our own java application.

2 . ESTADO DEL ARTE

2.1 CONCEPTOS DE SDN

2.1.1 INTRODUCCIÓN A LAS REDES DEFINIDAS POR SOFTWARE

El control distribuido y los protocolos de red de transporte ejecutando en routers y switches son los puntos clave que permiten que la información viaje por todo el mundo a través de internet. A pesar de estar ampliamente extendidas, las redes IP tradicionales son complejas y difíciles de administrar. Para satisfacer el alto nivel de las nuevas políticas de red, los operadores necesitan configurar cada dispositivo de manera individual utilizando comandos a bajo nivel y muy a menudo específicos de cada proveedor. Además de la complejidad, los entornos de red deben ser capaces de sobreponerse a posibles fallos realizando los pertinentes balanceos de carga. Sin embargo, en la actualidad no existen mecanismos de reconfiguración automática en las redes IP.

Por si no fuera poco, en las redes actuales nos encontramos que el plano de control (el cual decide cómo manejar el tráfico de red) y el plano de datos (el cual reenvía los paquetes recibidos en función de las decisiones del plano de control) están integrados dentro de los propios dispositivos de red. Esto reduce la flexibilidad y acota las posibilidades de evolución de las infraestructuras de red. Podemos ver un claro ejemplo en el paso de IPv4 a IPv6, una actualización que comenzó hace una década y hoy en día aún no se ha completado. Se puede observar por lo tanto, que era necesaria una reestructuración de la arquitectura para permitir agilizar estos procesos. Las Redes Definidas por Software (Software-Defined Networking o SDN) surgen como un paradigma alentador para poder afrontar las limitaciones expuestas anteriormente.

En primer lugar, rompe con la integración vertical separando el cerebro de la red (plano de control) del brazo ejecutor (plano de datos). En consecuencia, los switches se convierten en un dispositivo cuya única función se limita al reenvío de paquetes, y dónde el control de la red está centralizado favoreciendo las políticas de actualización, configuración y evolución de la red.

En la figura 5 podemos observar una vista simplificada de una arquitectura SDN. La separación de los planos de datos y de control puede ser realizada por medio de una interfaz de programación de aplicaciones (API). El ejemplo más extendido para la comunicación entre el controlador y los dispositivos de red es OpenFlow(OF). Un switch OF tiene una o más tablas de flujo donde almacena las diferentes acciones a realizar para los paquetes recibidos. Dependiendo de las indicaciones del controlador a un switch OF, este puede actuar como router, switch, firewall u otro tipo de tareas (por ejemplo: balanceador de carga, catalogador de paquetes, etc.) .

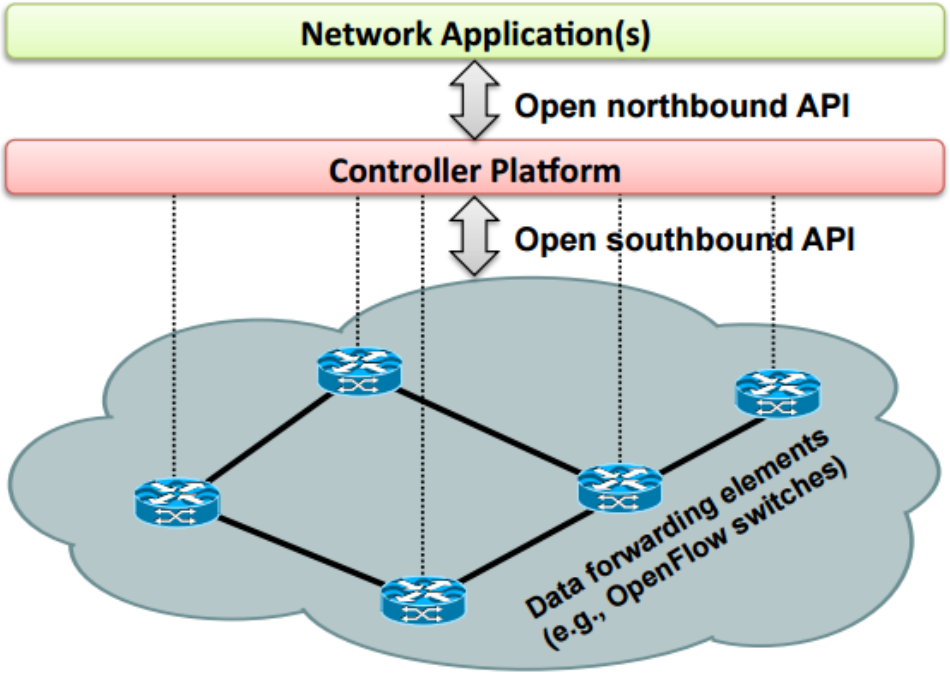


Figura 9: Vista simplificada de una Arquitectura SDN [8]

Una consecuencia a tener en cuenta en las redes SDN, es la separación de los problemas introducidos entre la definición de las políticas de red, su implementación física y el reenvío de paquetes. Esta separación es un punto clave en la buscada flexibilidad, fracturando el problema del control en pedazos manejables de manera independiente, haciendo más fácil la creación e introducción de nuevos elementos y simplificando el manejo, mantenimiento y evolución de la red.

Como todo proyecto, tanto SDN como OpenFlow empezaron siendo un experimento académico, pero debido a su potencial y a la importancia que ha adquirido en los últimos años, la mayoría de proveedores de dispositivos de red se han visto obligados a dar soporte de la API de OpenFlow en sus equipos. Tal es el empuje que trae detrás de sí dicha tecnología, que grandes empresas como Google, Facebook, Microsoft, Verizon o Deutsche Telekom, decidieron fundar la Open Networking Foundation (ONF) con el objetivo principal de promover el desarrollo y la evolución de las redes definidas por software basándose en estándares abiertos. A medida que las preocupaciones iniciales sobre la escalabilidad de las SDN han sido resueltas, ha pasado de ser un mero ejercicio académico a una solución comercial exitosa.

A modo de ejemplo, podemos ver como empresas de gran envergadura como Google, han desarrollado un entorno SDN para interconectar sus centros de datos a lo largo de todo el mundo. Otro ejemplo de la implementación de las soluciones SDN es la plataforma de virtualización NSX de VMware. [8]

Así pues, SDN ha generado un alto grado de interés entre la comunidad IT. A continuación expondremos brevemente algunos de los motivos por los que esto está sucediendo:

Las SDN facilitan la virtualización de servidores y las redes en la nube

Tradicionalmente, las organizaciones han solucionado con propuestas sencillas los problemas ocasionados por un aumento en la demanda de ancho de banda o capacidad de datos. Esto viene ocasionado por el alto coste de adquirir hardware adicional. Desafortunadamente, (y cuando no queda más remedio) la mayoría de corporaciones no pueden afrontar el coste de semejante inversión, especialmente teniendo en cuenta el crecimiento exponencial de la demanda. Sin embargo, la extrema competitividad de los mercados no permite afrontar las exigencias sin solventar dicho problema.

Uno de los agentes principales que permitió agilizar dicho proceso, fue la aparición hace aproximadamente 10 años de los primeros hipervisores (en inglés "hypervisors") o monitores de máquina virtual. Este tipo de tecnología fue desarrollada y lanzada al mercado inicialmente por la empresa VMWare. Esta permitía que un equipo utilizando un sistema operativo (independientemente del mismo), fuera capaz de ejecutar uno o más clientes con su propio sistema operativo, los cuales no necesariamente tenían que coincidir.

Lo que hizo VMware fue diseñar un pequeño programa que creaba un entorno virtualizado con el objetivo de sintetizar un entorno real de computación (NIC, BIOS, adaptador de sonido, de video de red,…), y, lo que es más relevante, permitía almacenar la información en un archivo que podía ser modificado, copiado o movido a otra máquina para su ejecución. Incluso, facilitaba pausar el SO sin que éste fuera consciente. Así nació lo que se conoce hoy en día como virtualización de sistemas operativos. Un avance que hacía posible tener a la vez en un mismo equipo varios servidores ejecutando a la vez (Windows o Linux), con el sustancial ahorro que ello suponía, y que, como se indicaba previamente, podían ser pausados, redistribuidos, clonados o copiados (agilizando y facilitando los procesos de backup) de manera rápida, cómoda y eficiente. Era lo que a posteriori se conoció como el inicio una nueva era en el ámbito informático, la era de la computación elástica.

Esta flexibilidad a la hora de manejar y/o administrar los servidores independientemente de su ubicación geográfica, permitió a los operadores de red ser capaces de empezar a optimizar la localización de los recursos dentro de un data center, así como su utilización basándose en parámetros tales como la refrigeración y la energía de los equipos. Por consiguiente, un operador podía perfeccionar la distribución de la carga dentro de un data center, dejando en reposo los servidores sin usar, ahorrando así una considerable cuantía en el coste energético. De manera análoga, le otorgaba la posibilidad de mover o ampliar el espacio de almacenamiento, la capacidad de computación y/o los recursos de red bajo demanda.

No obstante, y a pesar del continuo incremento de la capacidad de las matrices de comunicación (o switch fabric) y la velocidad de los enlaces, los protocolos de comunicación permanecían anclados en el uso de IP, MPLS y las tecnologías móviles.

Aquí es donde uno de los beneficios de SDN hace acto de presencia. Las SDN ayudan a aprovechar la virtualización de servidores para aumentar la eficiencia de los recursos, reducir la complejidad de la red y simplificar los procesos IT manuales para implementar, administrar y ajustar aplicaciones y redes. Las redes definidas por software amplían significativamente la eficiencia de la red y brindan soluciones a los problemas de capacidad sin que suponga un coste muy elevado.

Las SDN giran principalmente en torno a la automatización basada en políticas

Con el crecimiento de las redes, su mantenimiento y administración se han convertido en un proceso mucho más complejo. En el modelo tradicional, esta complejidad implica que es necesario contar cada vez con más recursos IT para manejar procesos como el aprovisionamiento, la configuración y la corrección. En pocas palabras, generalmente se trata de procesos manuales, por lo que ampliar la red de decenas a centenares de nodos implicaba que alguien tenía que intervenir y configurar manualmente muchos más dispositivos.

Las SDN dan un giro radical a esa situación, ya que automatiza estos procesos de aprovisionamiento, configuración y corrección de errores vía software. En vez de ser un empleado el que físicamente configure cada dispositivo, SDN habilita la opción de programar los cambios en todos los dispositivos de manera simultánea mediante el envío de paquetes con actualizaciones de software. [9]

No todas las implementaciones de SDN son iguales y no existe una única forma de usarlas. A modo de ejemplo, la estructura de la Infraestructura centrada en aplicaciones (ACI) de Cisco incluye dispositivos de redes inteligentes que simplifican de manera considerable la automatización de los procesos de red y acelera sustancialmente los tiempos de distribución de aplicaciones gracias a procesos IT simplificados y automatizados.[6]

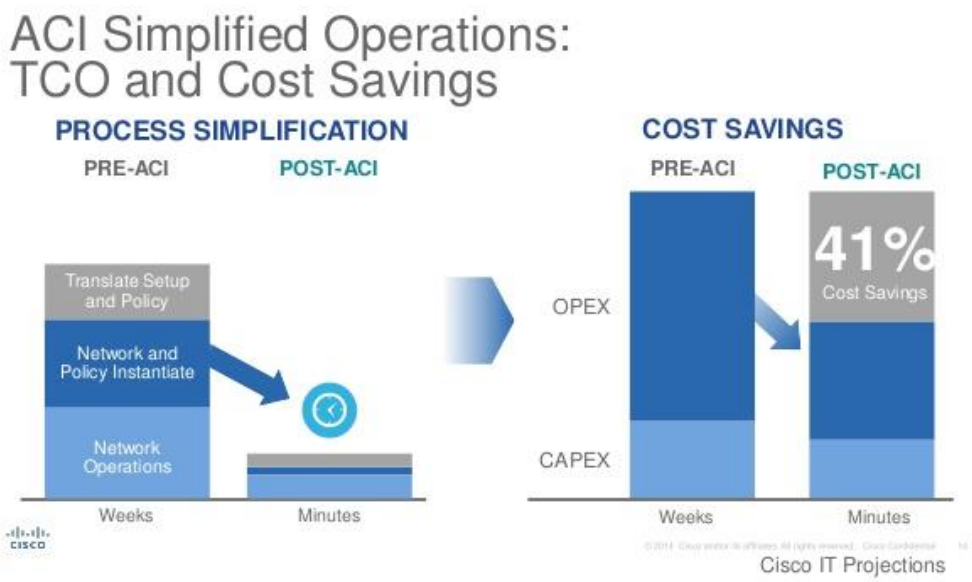


Figura 10: Ahorro de tiempo y costes de una arquitectura SDN

(fuente: http://www.cisco.com)

Ya hemos podido observar brevemente, el impacto que tendrán (y están teniendo) las redes definidas por software. Pero ahora surge otra cuestión, ¿cómo es la estructura de una arquitectura SDN?

El principal foco que permite a las SDN ser tan singulares, es que son dinámicas, fácilmente manejables, rentables a nivel económico y adaptable. Este tipo de arquitectura se caracteriza por dividir el panel de control de red y las funciones de reenvío de paquetes. Este (a priori) sencillo matiz, permite que el control de la red sea directamente programable y aislado del resto de capas, de manera que sea transparente para las aplicaciones y dispositivos de red.

Para comunicar todas estas capas, se usan APIS "northbound" (comunicación controlador - aplicación) y APIS "southbound" (comunicación controlador - dispositivos de red). El protocolo más utilizado en soluciones SDN para comunicar la capa de control con los elementos de red es Openflow, un protocolo de estándares abiertos mediante el cual el controlador indica a los switches las acciones a realizar sobre los paquetes entrantes.

2.1.2MODELO CENTRALIZADO VS MODELO DISTRIBUIDO

Los planos de control, datos y administración son el núcleo de las comunicaciones IP a la hora de poder transmitir un paquete con éxito desde un punto A, a un punto B de la red con una alta probabilidad de éxito. Estos planos son los cimientos sobre los que se sustentan las arquitecturas por capas (ej.: OSI), gracias a las cuales las redes han evolucionado hasta lo que son hoy en día. [8].

El primer punto a tratar sobre este tema, es la separación del plano de control y el plano de datos. Esto no es una idea nueva o revolucionaria, pero las redes definidas por software sí que aportan una nueva visión de cómo implementarlo.

Existen posturas encontradas respecto a cómo abordar dicha separación. Por un lado, hay quienes apuestan por una remodelación total de la red tal y como la conocemos hoy en día, promoviendo una centralización total. Y, por otro, quienes,conscientes de las complicaciones y el tiempo de adaptación que requiere un cambio de tal magnitud, apuestan por un modelo mixto. Modelo donde convivan las redes tradicionales con las SDN, e introduciendo un elemento ya conocido en otros protocolos de enrutamiento: el "router frontera". Este tipo de routers serían los encargados de hacer de intermediarios entre los diferentes tipos de redes, manteniendo unas independientes de otras. Sin embargo, para entender profundamente la importancia de la separación de dichos planos es necesario profundizar en el funcionamiento de cada uno de ellos.

PLANO DE CONTROL : "EL CEREBRO"

Cuando hablamos de plano de control, nos referimos a la parte lógica implementada en cada router que le permite saber cómo interactuar con sus vecinos, para crear las entradas de su tabla de reenvío. [10] Los datos utilizados para almacenar la topología de red, se conocen como información base de enrutamiento (RIB de sus siglas en inglés).

La RIB permanece actualizada de manera constante, mediante el intercambio de información entre los dispositivos de red (siguiendo las indicaciones de su plano de control). Las entradas en las tablas de reenvío, se generan a partir de la información base de reenvío (FIB de sus siglas en inglés). [9] Una entrada FIB se compone de la mínima cantidad de información necesaria para llevar a cabo una decisión de reenvío sobre un datagrama específico. [11] Se crean a partir de la información aportada por la RIB, una vez estas son estables y consistentes. Para ello, el plano de control debe obtener una vista de la topología de red que cumpla ciertos requisitos. La visión de la red que tiene cada dispositivo puede ser programada manualmente (rutas estáticas), aprendida mediante protocolos de enrutamiento (RIP, OSPF,...) o incluso una combinación de ambas.

En la figura 7 se muestra cómo se crea la FIB. Cuando un switch recibe un paquete en uno de sus puertos entrantes procedente de una dirección MAC desconocida, es redirigido al plano de control del dispositivo, donde se aprende, procesa y decide por qué puerto debe ser reenviado. Una vez que un paquete se ha enviado al plano de control, la información referente se almacena y puede generar una alteración de la RIB, con la consecuente transmisión de la información aprendida por todos sus puertos. Cuando la RIB se estabiliza, se actualiza la FIB tanto en el plano de control, como en el de datos.

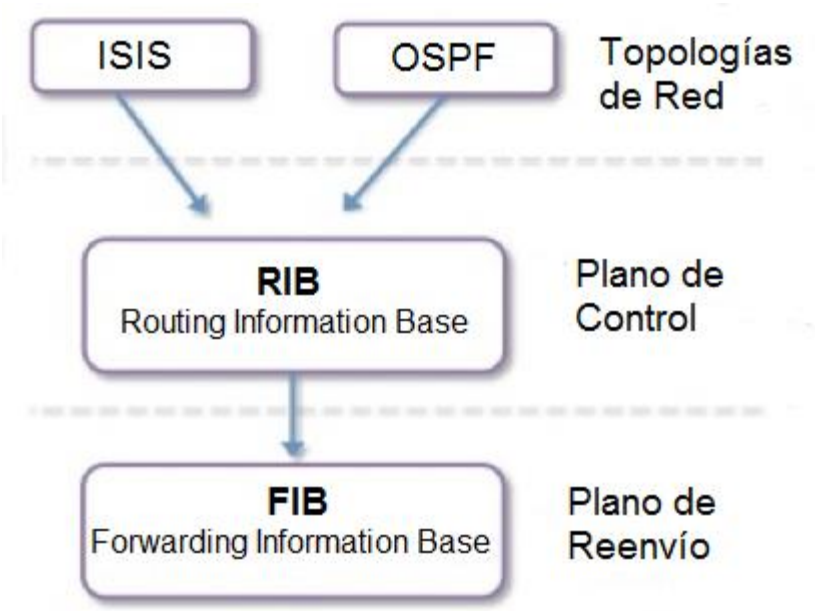


Figura 11: Creación de la Información Base de Reenvío (FIB)

(Fuente: Elaborada para este TFG)

En la realidad, los dispositivos de red siguen un modelo OSI, y en consecuencia cada capa tiene su plano de control. Así pues, el escenario antes mencionado es una combinación de los protocolos de control de las capas 2 y 3. El plano de control de capa 2 se centra en la parte hardware (direcciones físicas o MAC) y el de capa 3 a la comunicación a nivel de red (IP). [9]

PLANO DE DATOS

El plano de datos se encarga de procesar los datagramas entrantes utilizando una serie de operaciones de la capa de enlace: recogen el datagrama y realizan una serie de comprobaciones para asegurarse de que no ha sido corrompido durante la transmisión. Si el datagrama cumple los requisitos, se procesa en el plano de datos y mediante búsquedas en la tabla FIB se toma una decisión. Así se delimita la ruta más rápida para el paquete procesado. Cuando no se produce ninguna coincidencia, el datagrama se pasa al plano de control para que lo procese utilizando la RIB.

Las acciones más comunes resultantes de las búsquedas llevadas a cabo por el plano de datos en las tablas de reenvío son: reenviar el paquete (y en casos especiales como broadcast o multicast replicarlo) , descartarlo, encolarlo, contarlo o remarcarlo.

Algunas de estas acciones pueden ser combinadas o encadenadas. En ocasiones, se decide que se reenvien los paquetes al puerto local, lo que indica que el tráfico va dirigido a un proceso ejecutándose en el dispositivo, como puede ser en el caso de OSPF o BGP. Estos datagramas se envían utilizando un canal interno de comunicaciones.

Adicionalmente, el plano de datos puede implementar una serie de funcionalidades como listas de control de acceso (ACL de sus siglas en inglés) o políticas de calidad de servicio (QoS). Con este tipo de funcionalidades es posible alterar o adelantarse a la decisión de reenvío que tomará el plano de datos.

Este tipo de funcionalidades difiere con lo que un plano de datos debería ser, ya que este debería limitarse a sus funciones básicas o como mucho a la reescritura de algunos parámetros de las cabeceras. [9]

POR QUÉ ES TAN IMPORTANTE SU SEPARACIÓN

Hay múltiples argumentos a favor de la separación de los planos de control y de datos. Un punto clave a tener en cuenta, es la independencia en cuanto a evolución y desarrollo de las políticas de red. Así, el software de control de la red puede evolucionar independientemente de la capacidad de los dispositivos. También, tenemos la posibilidad de poder controlar toda la red desde un programa de alto nivel, usando comandos más sencillamente entendibles por un usuario básico, a la par que podemos corregir errores y llevar a cabo tareas de depuración y mantenimiento más fácil y cómodamente. Otro punto a tener en cuenta, es el balanceo de forma dinámica, rápida y segura. Si queremos que un router A, reenvíe paquetes a un router B, para llegar desde un equipo H1 a otro H2, es tan sencillo como indicarlo mediante una llamada RPC (Remote Procedure Call). Mientras que en la arquitectura de red actual, tendríamos que modificar todos los routers en la red para que se determinara mediante el protocolo de enrutamiento la ruta que queremos.

Pero es que la separación de planos no es únicamente más simple y efectiva, sino que también es menos costosa. Si tomamos como ejemplo un centro de datos (data center) común, con una media de 200.000 servidores y un despliegue en abanico de 20, se necesitan aproximadamente 10.000 switches para interconectar todos los dispositivos. Teniendo en cuenta únicamente el equipamiento, un switch de un proveedor estándar ronda los 5.000 dólares estadounidenses, mientras que si eliminamos la parte lógica (separación de planos) su precio es de alrededor de 1.000 dólares. Esto supone un ahorro de cerca de 40 millones de dólares en cada uno de estos data centers. En grandes empresas, las cuales poseen varios centros de datos (algunas como Google, Yahoo! o Facebook tienen más de 10) , la separación de planos supone un gran ahorro en coste hardware, que pueden emplear en el desarrollo, mantenimiento e innovación de la infraestructura. [12]

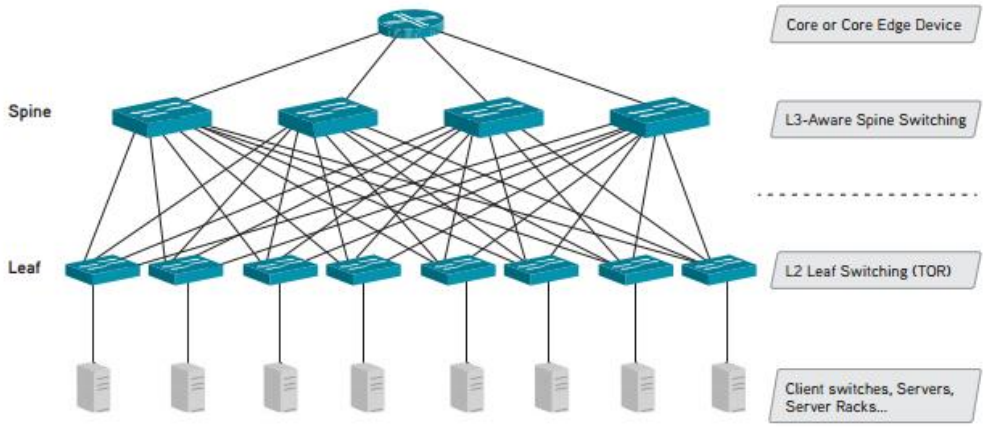


Figura 12: Arquitectura Data center (Despliegue en Ábanico) [12]

Sin embargo, la separación de los planos de control y de datos no es un concepto especialmente innovador. De hecho, existen routers con capacidad para múltiples ranuras independientes unas de otras. Una dedicada al proceso de control, y el resto encargadas de las funciones de reenvío de paquetes. En este tipo de dispositivos las tarjetas están conectadas usando enlaces internos de alta velocidad.

La principal novedad respecto a este enfoque que la aparición de las redes definidas por software aporta, es la implementación en grandes distancias (o al menos mayores a un metro). El plantear la idea de realizar este tipo de proyectos a grandes distancias y con un alto nivel de rendimiento, suponía un coste muy elevado y una alta complejidad con las tecnologías existentes. Aquí es donde SDN entra en escena con la idea de minimizar estos problemas. [9]

No obstante, también existen una serie de desafíos que las SDN deben afrontar para tener éxito. Al ser una estructura centralizada, nos encontramos con el problema de que un único controlador es el encargado de administrar una gran cantidad (a menudo miles) de dispositivos encargados de reenviar el tráfico de red, con el consiguiente problema de escalabilidad en el que ello deriva. Por otro lado, surge el desafío de ser capaz de mantener una alto grado de fiabilidad y seguridad. Al existir un único "gran cerebro" se genera un punto único de fallo, que en caso de no funcionar correctamente, puede comprometer el funcionamiento de toda la red. [12]

Así pues, podemos concluir que los planos de control distribuidos en sus diferentes formas, han evolucionado a lo largo del tiempo con la intención de satisfacer las preocupaciones de los operadores de red sobre la consistencia y la rápida convergencia de la red. Sin embargo, el plano de control distribuido no ha conseguido superar con éxito las barrreras de la flexibilidad de usuario, programabilidad o el alto nivel de dependencia e integración de los planos de control, datos y administración.

Por tanto, la idea de centralizar el plano de control de una manera lógica, pero físicamente distribuido, ha cobrado sentido desde la perspectiva de escalabilidad, alta disposición y visión geográfica.

2.1.3 ARQUITECTURA SDN

En este apartado, se pretende exponer una visión a grandes rasgos de los elementos que componen una arquitectura SDN. Dentro de la misma, podemos diferenciar tres planos: el plano de datos, el de control y el de aplicaciones.

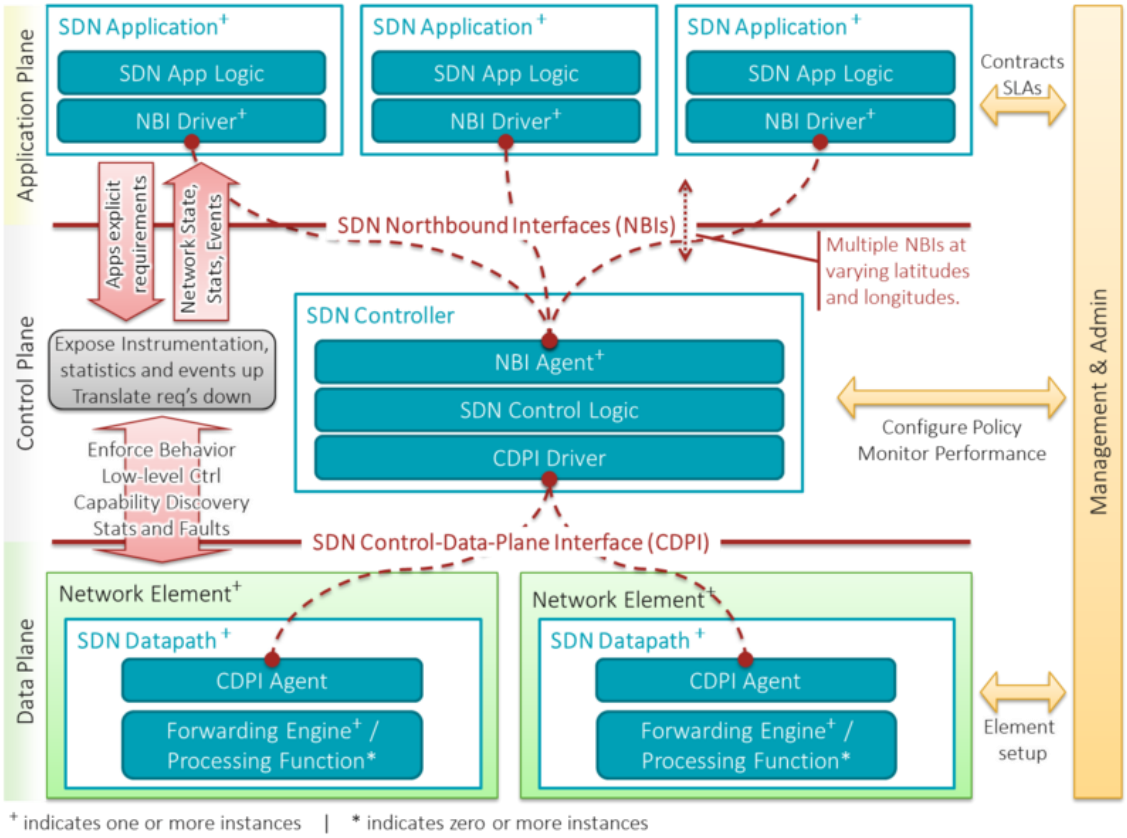


Figura 13: Arquitectura SDN Ampliada[13]

Como podemos ver en la figura 9, en la parte inferior tenemos el plano de datos, que abarca a los dispositivos de red cuyas rutas de datos SDN exponen sus prestaciones a través de su agente CDPI (interfaz entre el plano de datos y el plano de control o interfaz southbound (hacia el sur)).

Por otro lado, en la parte superior están las aplicaciones SDN, las cuales comunican al plano de control sus requisitos mediante una interfaz hacia el norte (Northbound Interface).

Entre medias de las dos, tenemos el controlador SDN. Este se encarga de "traducir" los requisitos demandados por las aplicaciones y administrar las rutas de datos de los dispositivos de red para que se cumplan.

En paralelo, tenemos el plano de administración y mantenimiento, responsable de la configuración de los elementos de red (asignando las rutas de datos) y de la definición de las políticas a llevar a cabo teniendo en cuenta los objetivos marcados por las aplicaciones y/o el controlador. Este tipo de arquitecturas está pensado para coexistir con redes que no sean del tipo SDN, con el objetivo de favorecer una migración de los modelos distribuidos antiguos al centralizado propuesto por las redes definidas por software.

Así pues, podemos concluir que dentro de una arquitectura SDN podemos diferenciar los siguientes componentes:

- **Aplicaciones SDN:** Cuando hablamos de aplicaciones SDN, nos referimos a programas que explícitamente comunican los requisitos a nivel de red necesarios para un óptimo funcionamiento, así como el comportamiento esperado dentro de la red. Estos requisitos son trasladados al controlador por medio de interfaces hacia el norte con sus respectivas APIs. Además, una aplicación SDN puede requerir una vista abstracta de la red para tomar sus propias decisiones internas.

- **Controlador SDN:** El controlador SDN es una entidad lógica centralizada, cuya misión es hacer de intermediario entre las aplicaciones y los elementos de red, preservando que se cumplan las peticiones demandados por las mismas. Un controlador SDN se compone de uno o más agentes encargados de manejar las interfaces hacia el norte (NBI), un driver CDPI y la parte lógica de control.

- **Rutas de Datos SDN (SDN Datapath):** Componente de red lógico, encargado de administrar y hacer visibles al resto las capacidades de reenvío y procesamiento del dispositivo. Se componen de un agente CDPI, un conjunto de uno o más motores de reenvío de tráfico y de funciones de procesamiento (pueden ser una, varias o ninguna). Las rutas de datos pueden contenerse en un único dispositivo (físico) de red y pueden estar definidas a lo largo de muchos de ellos.

- **Interfaz entre el Plano de Datos y el Plano de Control (CDPI):** Interfaz entre el controlador y la ruta de datos que permite (como mínimo) programabilidad de todas las operaciones de reenvío, información acerca de las capacidades del dispositivo, reporte estadístico y notificación de eventos.

- **Interfaz hacía el Norte (Northbound Interface):** Interfaces entre el controlador y las aplicaciones SDN que (generalmente) proporcionan una visión abstracta de la red, así como de su comportamiento y requisitos.

- **Gestión y Administración:** Este plano se encarga de las tareas estadísticas, que son manejadas al margen de los planos de datos, de control y de aplicación. A modo de ejemplo, podemos incluir dentro de estas tareas el manejo de la

relación proveedor-cliente, la asignación de recursos a clientes, la configuración de dispositivos físicos o la coordinación de la accesibilidad y los credenciales entre entidades físicas y lógicas. [13]

2.2 CONCEPTOS BÁSICOS DE VIRTUALIZACIÓN

Los hipervisores pueden clasificarse en dos tipos: aquellos que se ejecutan nativamente sobre hardware de metal base y, los que lo hacen sobre otro sistema operativo (el cual se ejecuta sobre el metal base) [14].

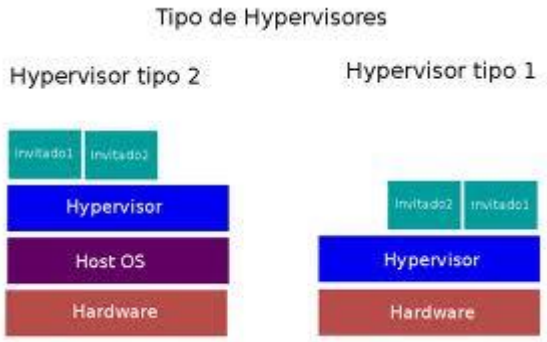


Figura 14: Tipos de Hipervisores

(fuente http://www.geocities.ws)

Dentro de los hipervisores de tipo 1, podemos encontrar algunos tan famosos como KVM (un hipervisor cuya infraestructura de virtualización reside en un núcleo Linux en hardware x86), Citrix o VMWare ESX (plataforma de virtualización compatible con arquitecturas x86 de pago producida por VMWare Inc.). Entre los de tipo 2, podemos destacar Virtual Box (software de virtualización propiedad de Oracle, y compatible con arquitecturas x86/amd64 y las principales distribuciones de Windows, Linux o MAC OS), VMWare: Workstation (versión de escritorio de pago y compatible con el mencionado ESX) o QUEMU(equivalente al Workstation pero para KVM). [14]

2.2.1 VIRTUALIZANDO LA RED

Cuando se aplica la virtualización a la red (NV de sus siglas en inglés), se crea una vista lógica basada en software del hardware y software de los recursos en la red (routers, switches, etc.).

Los dispositivos físicos se encargan simplemente del reenvío de paquetes, mientras que la red virtual (software) proporciona una abstracción lógica que facilita la implementación y administración de los servicios de red y sus recursos subyacentes. [12]

Son varias las soluciones vía software que nos facilitan tener un entorno de red virtualizado. A continuación introduciremos algunas de las más significativas y utilizadas durante el desarrollo:

- **OPEN VIRTUAL SWITCH:** OVS es un software de código libre que simula un switch multicapa. Proporciona una simulación virtual de un switch de producción de alta calidad, capaz de dar soporte a la mayoría de interfaces de

administración estándar y permitiendo hacer un uso libre de las funciones de reenvío por medio de su programabilidad. Soporta una gran cantidad de tecnologías de virtualización entre ellas Xen/Xen Server, KVM o Virtual Box. También permite crear switches del tipo OpenFlow para dar lugar a escenarios SDN manejables de manera fácil y eficaz.

- **MININET:** Mininet es un emulador de redes que permite disponer de equipos, switches, controladores y enlaces de tipo virtual. Los equipos virtualizados ejecutan un software de red Linux estándar y los switches son compatibles con el protocolo OpenFlow. Mininet es una herramienta muy utilizada para investigación, aprendizaje, desarrollo, testeo y muchas otras tareas con la ventaja de disponer de una red experimental en un único portátil o PC.

2.3 EL PROTOCOLO OPENFLOW

2.3.1 CONOCIENDO EL MUNDO DE OPENFLOW

Cuando hablamos de Openflow (OF), nos referimos a una interfaz de comunicaciones estándar entre las capas de control y de reenvío dentro de una arquitectura SDN. Esta, permite la manipulación directa del plano de control de los dispositivos de red (switches, routers, etc.), ya sean dispositivos físicos o virtuales. [15]

Sin embargo, en sus orígenes Openflow fue concebida como parte de un proyecto de investigación de la Universidad de Stanford. Cuando nació, lo hizo con la idea de permitir la creación de nuevos protocolos experimentales en la red interna del campus, que más tarde podrían ser usados para investigación o escenarios de test. Así, lo que surgió siendo una idea, evolucionó dando una visión de cómo, usando OpenFlow, se podía obtener un escenario igualmente funcional al proporcionado por los protocolos de enrutamiento de las capas 2 y 3. Este enfoque se conoce como la "proposición de borrón y cuenta nueva", y tiene como objetivo reinventar la manera en la que funcionan las redes tradicionales.

En 2011, se crea la Open Networking Foundation (ONF), un organismo sin ánimo de lucro formado por un grupo de proveedores con el objetivo de promover, estandarizar y comercializar el uso de OpenFlow en redes de producción. [14] Entre las empresas que forman parte de la ONF se encuentran algunas tan importantes como Google, Cisco, Microsoft, Facebook, HP, Ericsson, Oracle, Vodafone,... y un largo etcétera que abarca a más de 90 miembros incluyendo instituciones académicas y gubernamentales, empresas, proveedores de servicios, compañías de software y fabricantes de dispositivos. [16]

Los puntos clave que componen una arquitectura OpenFlow están relacionados con los ya comentados para un escenario básico de SDN:

- Separación de los planos de datos y de control (en el caso de la ONF, el plano de control se administra desde un sistema lógico centralizado).

- Utilización de un protocolo estándar de comunicación entre el controlador, y cada uno de los dispositivos de red a los que está conectado para asignarles un estado (en caso de Openflow el estado de reenvío).

- Proporcionar programabilidad a la red desde una vista centralizada utilizando una API moderna y extensible.

Openflow se compone por tanto de una serie de protocolos y una API. No es únicamente un producto en sí mismo. Por ello, los diferentes protocolos se pueden dividir en dos tipos:

- **Protocolos de conexión:** Se encargan del establecimiento de sesión , de la definición de las estructuras necesarias para las modificaciones de flujos, de almacenar estadísticas y de la creación y definición de los puertos y tablas de reenvío del switch.

- **Protocolos de configuración y mantenimiento:** Usan modelos de datos del tipo Yang y están basados en NETCONF. Se utilizan para garantizar una alta disponibilidad, para asignar puertos físicos del switch a un controlador determinado y para definir el comportamiento de la red en caso de fallo en el controlador. [9]

2.3.2 SWITCH OPENFLOW

Los fundamentos básicos de un switch OpenFlow son sencillos: un router convencional dispone de tablas de flujo que consulta para implementar funciones de NAT, QoS y reunir información estadística. No obstante, aunque guardan similitudes, dependiendo del proveedor el formato o características de las tablas puede variar. OpenFlow propone aunar todas esas cualidades en común para explotarlas y sacar beneficio con su uso, ya que proporciona un protocolo abierto capaz de programar las tablas de flujo en diferentes switches y routers.

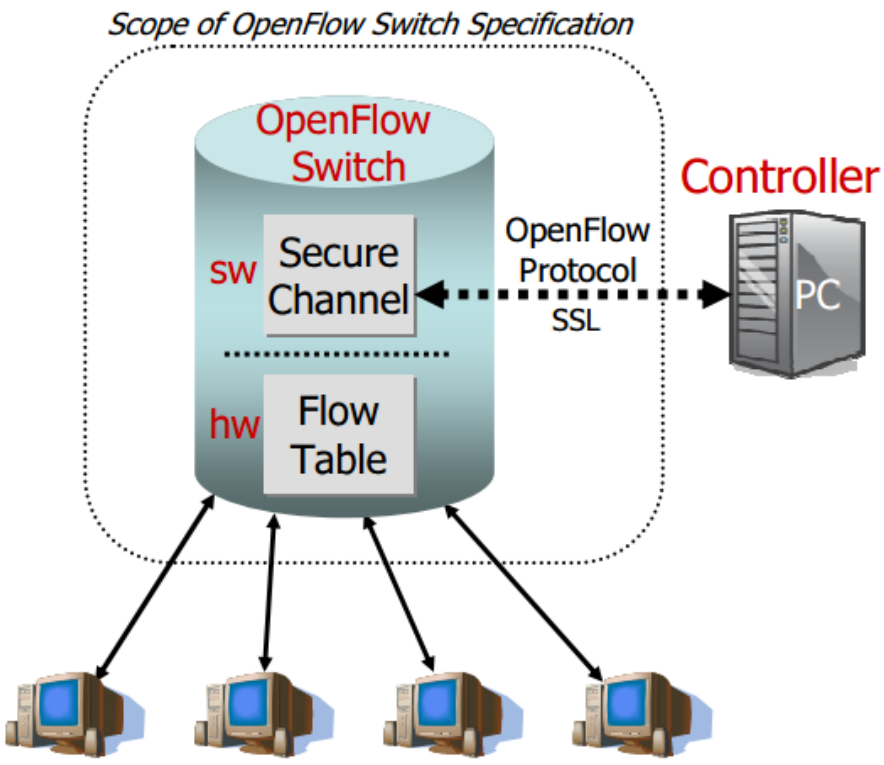


Figura 15: Partes de un Switch OpenFlow [17]

Dentro de un switch OpenFlow, podemos diferenciar al menos tres elementos:

- Una tabla de flujo, donde para cada entrada hay una acción asociada que indica cómo procesar el flujo de datos.
- Un canal seguro que conecta el switch con el controlador.
- El protocolo OpenFlow, que permite la comunicación entre switch y controlador.

La cantidad de acciones que se pueden realizar es muy amplia, pero todos los switches deben cumplir con unos requisitos mínimos. Toda entrada en la tabla de flujos debe tener tres campos: Una cabecera que identifique al flujo, la acción a realizar y una serie de estadísticas (número de paquetes y bytes de cada flujo y el tiempo que ha pasado desde que coincidió el último paquete con dicha entrada). Cada entrada puede tener adicionalmente una prioridad preestablecida, en caso de que haya más de una posible coincidencia para un mismo flujo de datos se ejecuta la acción de mayor prioridad. [17]

El protocolo OpenFlow ofrece la posibilidad de trabajar con puertos físicos, lógicos o reservados. Estos se usan como puertos de entrada, salida e incluso en modo bidireccional. A continuación se puede ver una breve descripción de los tipos de puertos reservados para la versión 1.3.3 de OF (algunos de estos puertos son requisito indispensable de un switch OpenFlow):

- **ALL (obligatorio)**: Se utiliza para enviar un paquete por todos los puertos del switch (sin incluir los reservados) a excepción de los configurados con la opción OFPPC_NO_FWD.

- **CONTROLLER (obligatorio):** Representa el canal seguro de comunicación en sentido bidireccional con los controladores OF. Cuando se usa como puerto de salida, se encapsula el paquete en un mensaje de tipo PACKET_IN. Cuando se usa como puerto de entrada recibe mensajes de tipo PACKET_OUT desde el controlador.

- **TABLE (obligatorio):** Este puerto representa el inicio de la cola OpenFlow. Se utiliza para entregar mensajes a la primera tabla de flujo de manera que el paquete pueda ser procesado según el algoritmo regular.

- **IN_PORT (obligatorio):** Puerto de entrada convencional.

- **ANY (obligatorio):** Puerto especial utilizado cuando no se indica específicamente ninguno. Puede ser utilizado tanto como puerto de entrada como de salida.

- **LOCAL (opcional):** Representa a la pila local y a la de administración del switch. Permite a entidades externas interactuar con el switch y sus servicios de red.

- **NORMAL (opcional):** Representa la parte tradicional de un switch normal (no OpenFlow). Se encarga de procesar los paquetes que le llegan usando el procedimiento habitual en un router convencional.

- **FLOOD (opcional):** Puerto de salida que se utiliza generalmente para reenviar el paquete por todos los puertos (sin incluir los reservados) a excepción del puerto por el que se recibió y los habilitados con la opción OFPPS_BLOCKED. [18]

La migración del modelo antiguo a (cómo se pretende en un futuro) un modelo basado en OpenFlow, conlleva un largo periodo de tiempo. Debido a que hay cientos de miles de routers y switches en el mundo actual. Entre las posibles opciones se pueden diferenciar dos tipos de switches OF:

- **Switches OpenFlow Dedicados:** Estos switches son creados específicamente para el uso de OpenFlow y no son compatibles con las capas 2 y 3 genéricas. Este tipo de dispositivos no tienen plano de control integrado y se limitan procesar los paquetes en función de sus tablas de flujo. Bajo este contexto, los flujos se definen a grandes rasgos y únicamente están limitados por las prestaciones de cada equipo.

  Como antes comentábamos, un switch OpenFlow debe ser capaz de llevar a cabo una serie de acciones mínimas. Así pues, podemos diferenciar tres tipos de acciones básicas: debe ser capaz de (consultando sus tablas de flujo) enrutar los paquetes entrantes por el puerto correspondiente, de encapsular un paquete y reenviarlo al controlador a través de un canal seguro y debe tener la posibilidad de descartar un flujo ya sea por motivos de seguridad, para frenar posibles ataques a la red o para evitar una posible congestión causada por mensajes broadcast de descubrimiento, procedentes de equipos finales utilizando otros protocolos de enrutamiento.

- **Switches Habilitados con OpenFlow:** En el otro lado, tenemos los modelos híbridos. Son switches convencionales a los que se les ha añadido la tabla de flujo, un canal seguro y el protocolo OpenFlow como una funcionalidad más.

  En este tipo de switches la tabla de flujo reutiliza hardware ya existente, como el caso de las TCAM. El canal seguro y el protocolo se portan para ser ejecutados por el sistema operativo del switch.

En este tipo de escenarios se tiene como objetivo aislar el tráfico experimental (procesado por la tabla de flujo) del tráfico genérico (o de producción) que es procesado por las capas 2 y 3 del switch. En la práctica hay dos formas de llevar a cabo con éxito este aislamiento: La primera, es utilizar el protocolo OpenFlow para añadir una nueva acción a la tabla de flujo. Esta acción indica que para el flujo de paquetes que coincida con dicha entrada, se pase a la cola de procesamiento habitual del router. La otra alternativa, es definir diferentes grupos de VLANs para el tráfico de experimentación y de producción. [17]

2.3.3. FUNCIONAMIENTO

Para cada paquete que llega a un switch OpenFlow se llevan a cabo una serie de acciones dentro del plano de control que determinan qué hacer con él.

Cuando se recibe el paquete en uno de los puertos (ya sea físico o virtual) lo primero que se hace es almacenar la información del puerto entrante para utilizarla en el procesado del paquete. Cada paquete viene con un pequeño conjunto de metadatos conocido como llave, que incluye diferentes campos (valores de la cabecera, puerto de llegada, ubicación de la carga de datos, etc.). Así pues, se extrae esta información y se pasa al resto de la pila de procesado.

A continuación, se usa la llave para encontrar la tabla correspondiente al paquete recibido. En caso de haber más de una (escenario habitual) se selecciona la primera tabla por defecto. Dentro de la tabla seleccionada, se utiliza la información obtenida de la llave, para encontrar una entrada de flujo que coincida (siempre se elige la primera coincidencia ya que es la de mayor prioridad) y se aplican las acciones asociadas a esa entrada de flujo. Llegados a este punto, si no se encontró ninguna coincidencia para la tabla seleccionada, se pasa a la siguiente (la cual debe tener un índice siempre mayor).

De haberse realizado la comprobación para todas las tablas de flujo del switch, se encapsula el paquete en un mensaje de tipo PACKET_IN y se envía al controlador para que sea él quien decida qué acción realizar sobre el mismo (salvo que el switch tenga ya una entrada que le permita descartar los paquetes para los que no se encuentra ninguna coincidencia).

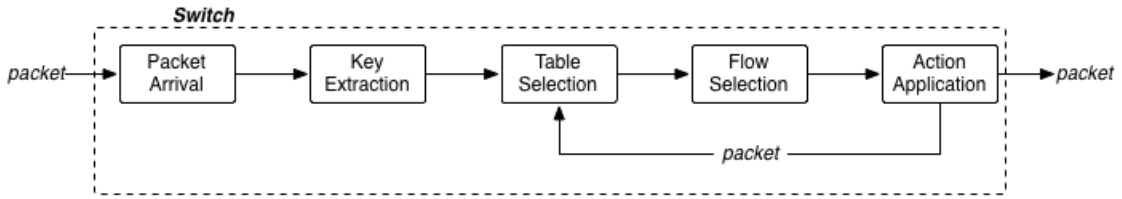


Figura 16: Lógica de un Switch OpenFlow [18]

2.3.4. TIPOS DE MENSAJES OPENFLOW

Dentro del protocolo OpenFlow podemos diferenciar tres clases principales de mensajes: del controlador al switch, asíncrono o simétrico.

De cada uno de ellos se pueden diferenciar subtipos, los cuales describimos a continuación:

Mensajes del Controlador al Switch

- ***Feature*:** El controlador solicita la identidad y las características básicas del switch mediante el envío de un mensaje "feature request".

- ***Configuration:*** El controlador puede establecer y/o consultar parámetros de configuración del switch.

- ***Modify-State:*** Las principales funciones de este tipo de mensajes son añadir, borrar, o modificar una o varias entradas de flujo en las tablas y establecer las propiedades de los puertos del switch.

- ***Read-State:*** Es utilizado por el controlador para consultar información del switch (configuración, estadísticas y competencias).

- ***Packet-out:*** El controlador utiliza estos mensajes para mandar paquetes de salida por uno de los puertos del switch y para reenviar los paquetes recibidos por medio de mensajes Packet-In.
  Todo mensaje Packet-out debe contener un paquete entero o un ID de buffer referenciando a un paquete almacenado en el switch. El mensaje también puede constar de una lista de acciones a ser aplicadas en el orden que se indica (si la lista de acciones llega vacía, se descarta el paquete).

- ***Barrier:*** Utilizado por el controlador para garantizar dependencias de mensajes o para recibir notificaciones de operaciones finalizadas.

- ***Role-Request:*** Se utiliza para establecer o requerir el rol en el canal OpenFlow.

- ***Asynchronous-Configuration:*** Se utiliza para establecer filtros adicionales en los mensajes asíncronos que quiere recibir por el canal OpenFlow.

Mensajes Asíncronos

- ***Packet-in:*** Con este tipo de mensajes se pasa el control de un paquete al controlador. Se pueden configurar para almacenar los paquetes durante un tiempo. Los switches que no tienen almacenamiento interno, los que no están

configurados para almacenar eventos packet-in o los que tienen el buffer lleno, deben mandar el contenido del paquete completo.

- ***Flow-Removed:*** Informan al controlador de la eliminación de una entrada de la tabla de flujo. Estos mensajes se envían únicamente para las entradas con el flag OFPFF_SEND_FLOW_REM activo.

- ***Port-status:*** Informa al controlador de cambios en un puerto. Se espera que el switch mande este tipo de mensajes al controlador cada vez que se modifique la configuración de un puerto o su estado.

- ***Error:*** Utilizados para notificar problemas al controlador.

Mensajes Simétricos

- ***Hello:*** Mensajes intercambiados entre switch y controlador durante la conexión inicial.

- ***Echo:*** Se usan principalmente para verificar que un dispositivo permanece operativo. Pueden ser de tipo request o reply.

- ***Experimenter:*** Mensajes utilizados para probar nuevas funcionalidades en fase de investigación.[18]

2.4 CONTROLADORES SDN

Como venimos comentando anteriormente, en toda arquitectura SDN hay un controlador que es el "cerebro" de la red. En teoría, un controlador SDN debe ser capaz de realizar las funciones del plano de control en un sistema distribuido, además de administrar el estado de los routers.

2.4.1 CONCEPTOS BÁSICOS

Todo controlador debe tener una serie de cualidades en común. Así pues cuando hablamos de controlador SDN nos referimos a un sistema de software (o conjunto de sistemas) que proporciona:

- Una gestión eficiente del estado de red y la distribución y el manejo del mismo. En algunos casos, este estado puede requerir del uso de bases de datos donde se almacena información, tanto sobre los elementos de la red adheridos al controlador y software relacionado, como sobre las aplicaciones SDN.

- Un modelo de datos de alto nivel que escenifica las relaciones entre los recursos administrados, las políticas y otros servicios proporcionados por el controlador. Normalmente se suele utilizar el lenguaje de modelado YANG.

- Una API RESTful (Transferencia de Estado Representacional) moderna se introduce para ofrecer los servicios del controlador a una aplicación. Esta interfaz facilita la comunicación entre controlador y aplicaciones.

- Un protocolo basado en estándares para obtener el estado de la red impulsado por las aplicaciones de los dispositivos de red (ej.: Openflow).

- Una sesión de control segura sobre el protocolo TCP entre el controlador y los agentes asociados en los dispositivos de red.

- Un mecanismo de descubrimiento de dispositivos, topologías y servicios, así como un sistema de cálculo de rutas. [9]

2.4.2 TIPOS DE CONTROLADORES

El primer controlador SDN fue el conocido como NOX, inicialmente desarrollado por Nicira Networks al mismo tiempo que OpenFlow. En 2008, tras la adquisición de Nicira por parte de VMWare se liberó el código fuente de NOX, para que la comunidad SDN pudiera hacer sus propios experimentos.

De aquí surgieron varios proyectos que llevaron al desarrollo de nuevos controladores tanto privados (como el Google WAN Controller o ONIX) como de código abierto (como son los casos de POX o Beacon).

Los controladores de código abierto empezaron a surgir a principios de 2010. Beacon, uno de sus más representativos es un controlador Openflow basado en Java y que más tarde daría lugar al proyecto Floodlight, que fue puesto a disposición del público bajo una licencia Apache 2.0.

Con posterioridad, muchos proveedores como Cisco, HP, IBM, VMWare o Juniper decidieron dar el salto al mercado de los controladores con sus propios diseños. En sus inicios, los diseños de HP, Cisco e IBM utilizaban Beacon como base, sin embargo, han ido migrando poco a poco a OpenDaylight (controlador desarrollado por Linux Foundation). [19]

Dentro de los tipos de controladores, podemos diferenciar los controladores OpenFlow. Estos no son más que un controlador SDN que utiliza como API hacia el sur el protocolo OpenFlow para conectar y configurar los dispositivos de red para determinar la mejor ruta a seguir por el tráfico de una aplicación.

A nivel particular, los controladores OpenFlow crean un punto de control centralizado para supervisar a los componentes de red de tipo OpenFlow habilitados y aumentan la flexibilidad de la red eliminando los protocolos particulares de cada proveedor.

A la hora de escoger un controlador SDN, toda organización IT debe tener muy en cuenta con anterioridad las funcionalidades que necesita, o puede llegar a requerir, así como las líneas de mejora marcadas por los desarrolladores. A modo de ejemplo, imaginemos que una compañía necesita hacer uso del protocolo IPv6. Este no está integrado con la versión 1.0 de OpenFlow, sin embargo sí que lo está en la 1.3 . Como es entendible, esta organización IT deberá por tanto escoger un controlador compatible con el estándar 1.3 de OpenFlow (o superior). [20]

A continuación exponemos una breve descripción de algunos de los controladores disponibles (tanto de código abierto como comerciales):

- **NOX:** Fue el primer controlador SDN y es un sistema operativo de red que proporciona una visibilidad y un control sobre los switches OpenFlow de la red. Dispone de soporte para aplicaciones escritas en Phyton y C++.

- **BEACON:** Controlador OpenFlow extensible basado en Java y construido bajo una infraestructura OSGI. Esto permite que las aplicaciones se monten usando esta plataforma y puedan ser arrancadas, paradas, recargadas o instaladas en tiempo de ejecución sin necesidad de desconectar los switches.

- **FLOODLIGHT:** Floodlight surge como una evolución de Beacon, y aunque en sus comienzos utiliza la base de dicho controlador, termina por utilizar ApacheAnt (herramienta de montado software de la compañía estadounidense).

  Floodlight tiene una comunidad de usuarios muy amplia y una gran cantidad de funcionalidades que pueden ser añadidas. Pero además, dispone de un entorno gráfico basado en Java y otro web donde podemos encontrar la mayoría de sus funcionalidades y hacer uso de ellas por media de una API REST.

- **OPENDAYLIGHT**: OpenDaylight es un proyecto colaborativo de la Fundación Linux que ha recibido un gran apoyo por parte de importantes empresas (entre ellas Cisco o Big Switch).

  Al igual que Floodlight OpenDaylight está escrito en Java y tiene detrás de sí una de las mayores comunidades de usuarios. Así mismo, también dispone de una API REST, un entorno gráfico web y un sinfín de funcionalidades a añadir.

  Uno de los puntos que diferencia a OpenDaylight del resto, es que soporta otros protocolos hacia el sur además de OpenFlow.

- **BROCADE VYATTA:** Es un controlador basado inicialmente en la versión de OpenDaylight con el nombre de Helium anunciada en Septiembre de 2014. Incorpora a las funcionalidades básicas y la posibilidad de utilizar la orquestación OpenStack. Aunque es de código abierto, dispone también de una versión comercial.

- **CONTROLADOR BIG SWITCH:** Controlador de código privado basado en Beacon y que tiene como objetivo las redes de producción empresariales. Proporciona una interfaz de línea de comandos fácil de usar desde la que administrar la red.

- **CISCO ACI (Infraestructura de control de aplicaciones) :** La solución de Cisco está únicamente desplegada en alrededor de 9000 switches, los cuales pueden formar una arquitectura utilizando diversas tecnologías incluyendo IS-IS y VXLAN para calcular las rutas de reenvío. Otro de los aspectos diferenciales es que usa su propio protocolo hacia el sur (Cisco OpFlex), aunque también soporta el uso de OpenFlow.

2.4.3 FACTORES INFLUYENTES EN LA ELECCIÓN DE UN CONTROLADOR

La correcta elección del controlador para una arquitectura SDN puede ser un factor determinante para cumplir con los objetivos marcados. A continuación, se expondrán brevemente algunos de los factores más influyentes en la toma de esta decisión:

1. **Soporte OpenFlow:** Un administrador debe tener en cuenta las necesidades de su entorno de red y elegir un controlador compatible con una versión de OpenFlow que se adapte a la situación demandada.

2. **Virtualización de Red:** Soporte para virtualizar la red. Esta característica permite a los administradores crear redes virtuales basadas en políticas de forma dinámica.

3. **Funcionalidad de la Red:** Con el objetivo de obtener una mayor flexibilidad es importante que un controlador SDN disponga de la capacidad para realizar diferentes funcionalidades. Entre ellas podemos destacar: la toma de decisiones de enrutamiento basado en múltiples campos de la cabecera OpenFlow, capacidad para descubrir múltiples caminos desde el origen del flujo a su destino o para dividir el tráfico de un flujo dado a través de múltiples enlaces.

4. **Escalabilidad:** Dado que es el "cerebro" de la red, es importante tener en cuenta el número de dispositivos que un único controlador puede administrar de manera eficiente (se debe exigir un mínimo de 100 switches).

5. **Rendimiento:** Una de las funciones principales de un controlador es añadir flujos a las tablas de los switches. Por lo tanto, un factor a tener en cuenta es su capacidad de procesado (tiempo de conformación de un flujo y número de flujos por segundo que puede establecer).

6. **Programación de red:** Como podíamos ver en secciones anteriores, la programabilidad de la red es una de las características principales de las SDN. Es lógico por tanto, que se busque que nuestro controlador disponga de las herramientas necesarias para, entre otras funcionalidades, redirigir tráfico y aplicar filtros sofisticados a paquetes.

7. **Confiabilidad:** Un controlador SDN debe ser capaz de aportar fiabilidad y consistencia a la red. Para ello, almacena los posibles caminos desde un origen a un destino, de manera que si se produce un fallo en el enlace, sea capaz de redirigir el tráfico de manera rápida y eficaz.

8. **Seguridad de la red:** Para garantizar la seguridad de los flujos de datos, un controlador debe soportar autenticación y autorización, así como ser capaz de detectar posibles sospechas de ataques maliciosos.

9. **Monitorización centralizada y visualización:** En un entorno SDN es importante que el controlador SDN pueda tener una visión global de la red para determinar correctamente las acciones a realizar sobre los flujos de datos. También debe ser capaz de obtener estadísticas utilizando herramientas y protocolos para la monitorización de la red. La situación ideal sería que el controlador permita el acceso a información de la red (dispositivos conectados, estado de los puertos o estado de enlace) a través de una REST API.

10. **Fabricantes de controladores SDN:** Dado que las SDN son una tecnología reciente y que está suscitando un gran interés, es importante elegir un controlador que nos aporte unas ciertas garantías de que va a ir actualizándose a medida que evolucione el concepto de SDN. Por ello es importante elegir un controlador con el suficiente apoyo, ya sea económico o con una gran comunidad de usuarios detrás.

11. **Soporte de plataformas:** Los controladores SDN se ejecutan sobre diferentes sistemas operativos. Por lo tanto, siempre es un punto interesante disponer de un sistema multiplataforma que aumente la flexibilidad y la portabilidad de nuestro controlador.

12. **Procesamiento:** Evaluar la capacidad de un controlador para trabajar con procesos múltiples o no, puede repercutir en la escalabilidad de los núcleos de la CPU. No tendría sentido utilizar un controlador mono proceso en un hardware que posee múltiples CPU y viceversa. [21]

3. DISEÑO E IMPLEMENTACIÓN

3.1 INTRODUCCIÓN

Como se indicaba previamente en la introducción, el objetivo principal de este proyecto es el de elaborar un guion de prácticas. Para ello, será necesario el despliegue de una arquitectura SDN, así como la comprensión del entorno mediante la realización de una serie de pruebas y ejemplos en los que se basa el guion expuesto en el punto 5.

En consecuencia, se utilizará un esquema distinto en cada parte. No obstante, el objetivo es disponer de un entorno virtualizado que simule un escenario real, mediante el uso de máquinas virtuales independientes.

Por una sencilla razón de comodidad, durante las pruebas realizadas se ha utilizado un único host para lanzar todas las máquinas virtuales. Pero dado que son independientes, podrían utilizarse de manera intercambiable y en diferentes equipos.

Por lo tanto, se necesitará de tres tipos de máquinas virtuales básicas:

1. **Equipos (cliente/servidor):** Máquina virtual que simulará las funciones de un PC convencional y que utilizaremos como cliente o servidor. Esta máquina es ajena a todo tipo de protocolo relacionado con SDN, y en caso de necesitar comunicarse con otro equipo lo hará utilizando los mecanismo habituales (IP, TCP, ARP, UDP…).

2. **Switch:** Máquina virtual que ejercerá las labores de un switch OpenFlow. Para ello, esta máquina hará uso del software de OpenVirtualSwitch. Dicho switch, operará como un simple dispositivo de reenvío (no dispondrá de plano de control) dentro de la red, siguiendo las órdenes establecidas en su tabla de flujos por el controlador.

3. **Controlador:** Máquina virtual cuya función será ejercer el papel de controlador SDN dentro de la arquitectura desplegada (diríjase a la sección 2.3 para tener una visión detallada de las funciones de un controlador).

Para llevar a cabo el diseño, y a pesar de haber diferentes arquitecturas, se partió del objetivo de llegar a tener una estructura sencilla cuya máxima complejidad fuera la mostrada en la figura 13:

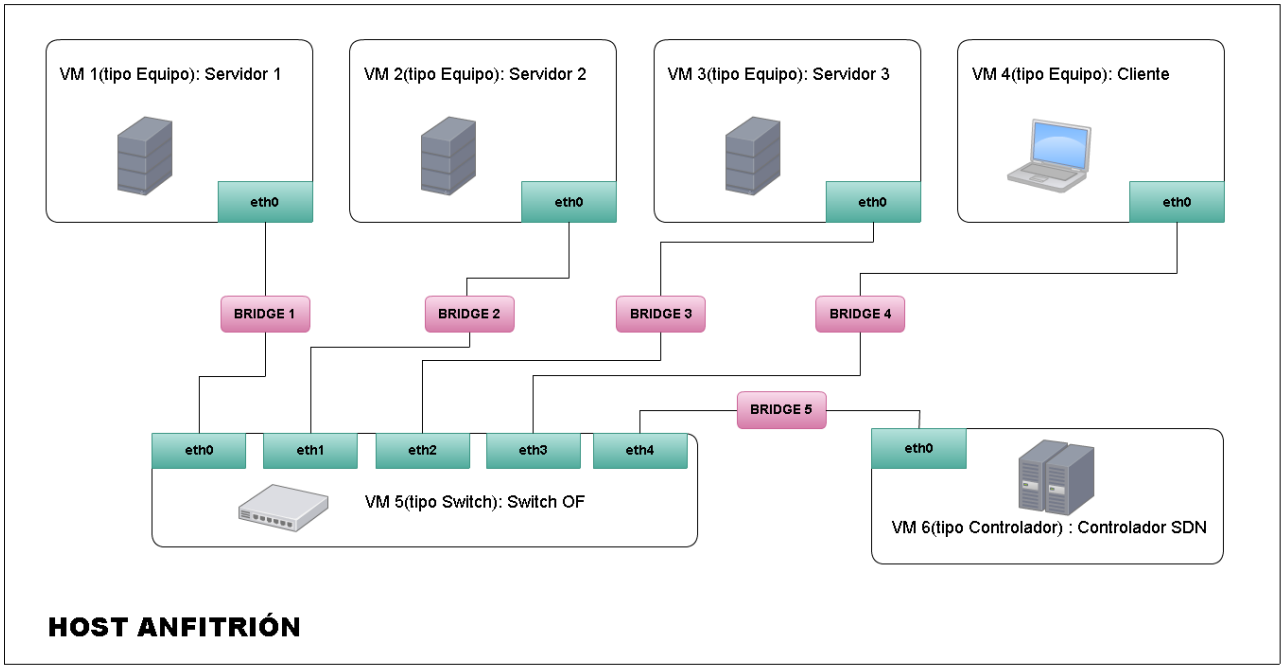


Figura 17: Esquema General de la Arquitectura Desplegada

(Fuente: Elaborada para este TFG)

3.2 DECISIONES DE DISEÑO DE LAS MÁQUINAS VIRTUALES

Dado el escenario planteado, la primera decisión a llevar a cabo era decidir qué sistema operativo deberían llevar las máquinas virtuales, y que hipervisor utilizar para ejecutarlas.

Como primer paso lógico se decidió utilizar una distribución Linux, ya que las herramientas utilizadas (OpenVirtualSwitch, OpenDayLight,…) han sido diseñadas y optimizadas para compatibilizar con este sistema operativo. En un primer momento, se optó por una distribución Ubuntu en su versión LTS (Long Term Support) pero tras varias pruebas en tiempo de ejecución, se comprobó que con demasiada frecuencia el rendimiento no era el adecuado cuando había varias máquinas virtuales ejecutando en paralelo. Por ello se optó por probar una distribución de Ubuntu más ligera: LUbuntu.

LUbuntu es una distribución Linux y cuyo núcleo principal está basado en Ubuntu, incluyendo únicamente una serie de herramientas esenciales, que le permiten ser mucho más rápido y menos pesado.

En cuanto al hipervisor, se ha utilizado Virtual Box pero otras alternativas como KVM son perfectamente compatibles.

En los siguientes puntos haremos un análisis más en detalle de las especificaciones de cada uno de los tipos de máquinas virtuales y el porqué de esas especificaciones.

3.2.1 MÁQUINA VIRTUAL TIPO 1: EQUIPO/HOST

Esta máquina virtual tendrá como sistema operativo LUbuntu como ya comentamos previamente y dispondrá de 256 MB de RAM. Al ser una máquina que simplemente hará las veces de cliente/servidor, se ha estimado oportuno que con dicha cantidad de memoria RAM será capaz de desempeñar sus funciones ampliamente. De esta manera, permitimos que al estar ejecutando todas las máquinas virtuales en paralelo una vez implementada la arquitectura, el host huésped no se sature.

Respecto a las interfaces, dispone de 2 interfaces de red: una de tipo NAT para instalación y actualización de aplicaciones y una segunda en modo bridge que será la que utilizaremos para conectarnos a las otras máquinas virtuales (al ser esta de tipo host, únicamente se conectará a una máquina de tipo switch).

3.2.2 MÁQUINA VIRTUAL TIPO 2: SWITCH OPENFLOW

Esta máquina virtual utilizará también LUbuntu, pero dado que sus funciones serán algo más exigentes que las de un simple host, hemos decidido aumentar su capacidad de memoria RAM a 512 MB.

Esta VM dispondrá de 5 interfaces para conectar el switch con los diferentes equipos y con el controlador. Sin embargo, no todas tienen por qué ser usadas en todos los escenarios necesariamente.

Por defecto, Virtual Box dispone de hasta 8 tarjetas de red disponibles por máquina virtual, sin embargo, sólo 4 se pueden habilitar desde el entorno gráfico de usuario. Para saber cómo hacer uso de las 4 restantes diríjase al apartado 3.3.2 donde se muestra cómo añadir una quinta interfaz al switch.

Adicionalmente, se dispone del software Open Virtual Switch instalado manualmente. Este permite simular un switch OpenFlow y su funcionamiento dentro de la máquina virtual. Para saber más acerca de OVS diríjase a la sección 2.2.1.

Por último, para saber cómo crear, añadir o configurar un nuevo switch lea el punto “Configuraciones Previas” del guion de prácticas elaborado.

3.2.3 MÁQUINA VIRTUAL TIPO 3 : CONTROLADOR SDN

Por último, tenemos la máquina virtual que ejercerá las funciones de controlador SDN.

El controlador que utilizaremos en este entorno es OpenDaylight. En la sección 2.4.3 de este documento se dispone de un apartado más detallado sobre cómo elegir el controlador más adecuado para cada arquitectura SDN.

Teniendo en cuenta estos parámetros y las diferentes alternativas se llegó a la conclusión de que OpenDaylight era el controlador que mejor cumplía los requisitos de este proyecto. Ya que el principal objetivo era desplegar una infraestructura SDN lo más parecida posible a un escenario real, y dado que ODL era uno de los controladores más apoyados por grandes empresas (Cisco, Microsoft, Dell, Ericsson, Intel y un largo etcétera) y con una de las mayores comunidades de usuarios y desarrolladores frente a sus competidores, se concluyó que era una de las alternativas con más posibilidades de convertirse en estándar en un futuro, factor que declinó definitivamente la balanza.

El S.O. elegido como en todas las máquinas es LUbuntu, pero con una memoria RAM de 2758 MB debido al alto rendimiento exigido por las tareas de procesamiento del controlador.

Se dispone de dos tarjetas virtuales de red: una de tipo NAT y otra en modo bridge para conectarla al switch Openflow. En caso de necesitar más en futuros escenarios (en los que hubiese más switches), estas pueden ser habilitadas.

Dentro de las máquinas virtuales de tipo 3, utilizaremos dos versiones: una primera que incorpora una versión básica de OpenDaylight para pruebas a nivel de usuario y una segunda que dispondrá de una versión de ODL para desarrolladores de aplicaciones SDN. Para este proyecto se ha utilizado la última versión disponible en la fecha que comenzó: con el nombre de Lithium (versión SR3), publicada el 3 de Diciembre de 2015.

Para conocer más detalles acerca de OpenDaylight puede dirigirse a la sección 2.4.4.

3.3 IMPLEMENTACIÓN Y DESPLIEGUE

Para llevar a cabo la implementación de las máquinas virtuales bajo las condiciones de diseño indicadas previamente, será necesario un proceso de configuración adicional en las máquinas disponibles.

Lo que se pretende explicar en los siguientes apartados es el proceso realizado para dejar dichas máquinas virtuales operativas, así como las operaciones necesarias para montar cada una de las diferentes infraestructuras sobre las cuales se realizarán las pruebas que dan lugar al guion de prácticas. No obstante, en el guion se dispone de una serie de instrucciones genéricas a tener en cuenta para el uso de cada una de las máquinas virtuales.

3.3.1 PREPARANDO LAS MÁQUINAS VIRTUALES

Antes de comenzar a desplegar las diferentes arquitecturas, cada una de las máquinas virtuales necesitaban de una pequeña puesta a punto. Para ello se generó una máquina virtual con el nombre de "LUbuntu Base" la cual sería el punto de referencia para cada nueva máquina virtual de la que no existiese aún una genérica de su tipo.

PC_XX

Esta máquina virtual ha sido diseñada siguiendo los patrones de diseño marcados en el apartado 3.2.1 y por lo tanto puede ser utilizada como cliente o servidor dentro de una topología de red.

Las operaciones llevadas a cabo para generar dicho tipo de máquina virtual, son sencillas y se explican brevemente a continuación:

1. Clonación "LUbuntu Base".

2. Configuración mediante la interfaz gráfica de usuario de Virtual Box de los parámetros de diseño establecido (memoria RAM e interfaces habilitadas).

3. Con la máquina arrancada se realizan funciones de actualización (comandos Linux `"apt-get upgrade" y "apt-get update".`

4. Instalación de las herramientas wireshark, iperf y curl disponibles mediante el comando `"apt-get install <nombre paquete>"`

OVS_XX

Esta máquina virtual ha sido diseñada siguiendo los patrones marcados en el apartado 3.2.2 y, por lo tanto, puede desempeñar las funciones de un switch OpenFlow dentro de una arquitectura SDN.

Las operaciones llevadas a cabo en un principio no difieren de la de la máquina anterior (PC_XX). No obstante, además de las ya indicadas, habría que proceder a la instalación del software de Open Virtual Switch.

Desde la misma web de OpenVSwitch nos ofrecen la oportunidad de descargar un archivo comprimido para su uso e instalación [22]. Pero desde cualquier distribución Ubuntu podemos instalarlo directamente con el comando `"apt-get install openvswitch*"`, el cual nos instalará todos los paquetes disponibles.

CONTROLADORES

Como ya comentamos previamente usaremos dos máquinas virtuales distintas para simular nuestro controlador SDN: una para el despliegue de las arquitecturas 3.3.3 y 3.3.4 donde usaremos las funcionalidades integradas en OpenDaylight (versión usuario) y otra que utilizaremos para añadir nosotros mismos una pequeña aplicación (versión desarrollador) en la arquitectura descrita en el punto 3.3.5.

Para la primera máquina virtual llevaremos a cabo los pasos indicados en el punto 3.3.1.1 (PC_XX) para tener una configuración inicial. A esto (como pasaba con el caso anterior) deberemos añadirle el software de OpenDaylight.

El proceso de instalación para la versión de usuario es sencillo:

1. Para utilizar OpenDaylight se requiere de una versión de JRE (Java Running Environment o Entorno de Ejecución de Java) de 7 o superior, dependiendo de la versión de ODL utilizada. Por lo que procederemos a su instalación:

    a. Instalación de los paquetes:

    ```
    sudo apt-get install openjdk-7-jdk
    ```

    b. Actualizamos las variables de entorno:

    ```
    sudo update-alternatives --install /usr/bin/java java
/usr/lib/jvm/java-7-openjdk-amd64/bin/java 1

    sudo update-alternatives --config java

    sudo nano ~/.bashrc

    export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
    ```

2. Descargamos el paquete desde la web de OpenDaylight [23] y lo descomprimimos en el directorio que deseemos.

    Llegados a este punto, disponemos de una versión del controlador en nuestra máquina virtual. Para lanzarlo debemos ejecutar el comando `"./bin/karaf"` desde la raíz del archivo descomprimido, y esperar hasta que aparezca un prompt con el texto "opendaylight".

Para la máquina virtual que nos permitirá añadir nuestras propias aplicaciones necesitamos, además del JDK, una versión de Maven 3.0 o superior. Dado que para el despliegue de la arquitectura utilizaremos una aplicación desarrollada por el grupo de SDNHub, hemos utilizado una máquina virtual preparada por ellos mismos y con

OpenDaylight ya instalado. Si se desea conocer más acerca de los detalles de instalación puede consultarse la guía para desarrolladores disponible en la web de OpenDaylight [23].

3.3.2 DESPLIEGUE DE ARQUITECTURAS

Dado que las arquitecturas utilizadas son muy similares, a continuación se explica cómo configurar un escenario compuesto por 4 equipos (un cliente y tres servidores), un switch OpenFlow y un controlador OpenDaylight (figura 13). El resto de escenarios utilizados en el guion son sub-arquitecturas de este, por lo que el procedimiento será equivalente quitando las máquinas virtuales que no necesitemos.

Para este escenario, utilizaremos 4 máquinas virtuales de tipo PC_XX que actuarán como hosts, una de tipo OVS_XX que desempeñará las labores de switch OpenFlow y la versión de usuario de OpenDaylight disponible en la máquina "ODL Basic Controller". Dado que se facilitan estas máquinas de manera genérica, para disponer de la arquitectura deseada será necesario configurar cada una correctamente.

En primer lugar procederemos a configurar las máquinas virtuales de tipo host. Se explicará el proceso a llevar a cabo para el PC_01, ya que es análogo al resto de equipos:

1. Clonamos la máquina virtual PC_XX renombrándola correctamente.

2. Se crea y se habilita un bridge en el host anfitrión que servirá para comunicar esta máquina virtual con la correspondiente al switch OpenFlow:

```
sudo brctl addbr ovs1-pc1
sudo ip link set ovs1-pc1 up
```

3. Configuramos la interfaz de red: Para ello en el menú de configuración de red de dicha máquina virtual habilitamos una interfaz en modo bridge y elegimos el creado en el paso 2. También es necesario habilitar el modo promiscuo y cambiar la dirección MAC por la elegida (00:00:00:00:00:01).

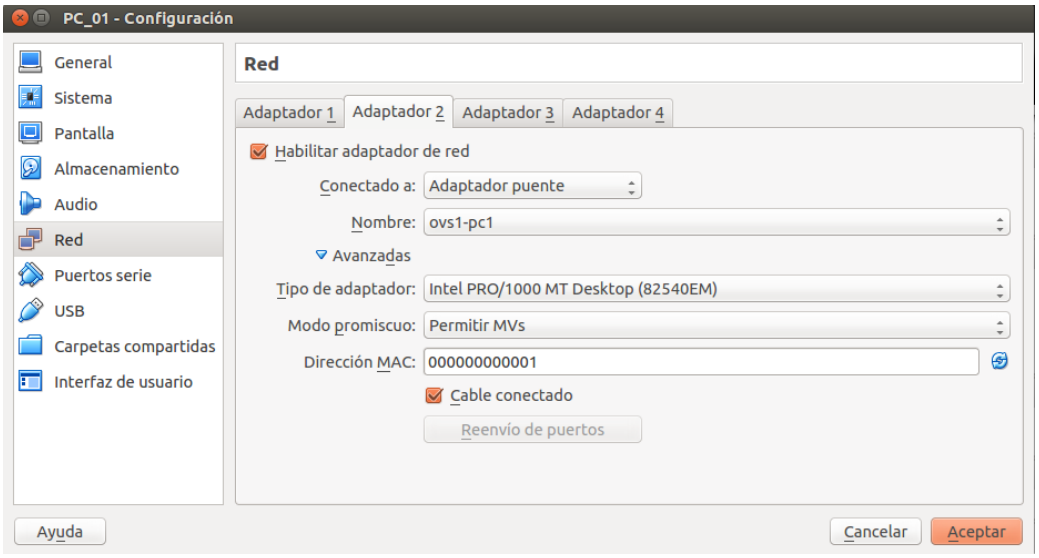


Figura 18: Configuración de una Interfaz en VirtualBox (GUI)

(Fuente: Elaborada para este TFG)

4. Configuración IP :

   a. Con la máquina arrancada, abrimos una terminal y editamos el fichero de interfaces de red ubicado en la ruta (<raíz>/etc/network/interfaces). Para añadir una dirección IPv4 que permanezca estática cada vez que arranquemos la máquina virtual debemos añadir el siguiente texto (la otra opción es hacerlo de forma manual cada vez que iniciemos la máquina con el uso de *"ifconfig"*) .Supongamos que queremos configurar en la interfaz enp0s8 del PC_01, la IP 100.14.1.1 perteneciente a la subred 100.14.1.0/24 :

```
auto enp0s8
face enp0s8 inet static
address 100.14.1.1
netmask 255.255.255.0
broadcast 100.14.1.255
```

   b. Reseteamos la configuración de red y comprobamos que se añadió correctamente.

```
sdn@sdn:~$ ifconfig
enp0s8    Link encap:Ethernet  direcciónHW 00:00:00:00:00:01
          Direc. inet:100.14.1.1  Difus.:100.14.1.255  Másc:255.255.255.0
          Dirección inet6: fe80::200:ff:fe00:1/64 Alcance:Enlace
          ACTIVO DIFUSIÓN FUNCIONANDO MULTICAST  MTU:1500  Métrica:1
          Paquetes RX:0 errores:0 perdidos:0 overruns:0 frame:0
          Paquetes TX:8 errores:0 perdidos:0 overruns:0 carrier:0
          colisiones:0 long.colaTX:1000
          Bytes RX:0 (0.0 B)  TX bytes:648 (648.0 B)
```

Figura 19:Comando Ifconfig

(Fuente: Elaborada para este TFG)

El procedimiento para configurar la máquina OVS_01 es análogo, pero con una serie de matices:

1. Clonamos la máquina virtual OVS_XX renombrándola correctamente.

2. Se presupone que ya se han creado los bridges en el proceso de configuración de las máquinas virtuales de tipo host, por lo que no es necesario crear más. Así pues, utilizaremos dichos bridges para habilitar 4 interfaces de red en modo adaptador puente (una por cada equipo) como se indicaba en el paso 3 de la configuración anterior.

3. Creamos el bridge que servirá de enlace entre el switch y el controlador:

```
sudo brctl addbr ovs1-odl1
sudo ip link set ovs1-odl1 up
```

4. Añadimos la última interfaz: Esta, se configura de manera distinta al resto ya que , aunque Virtual Box dispone de 8 tarjetas de red, no todas se pueden habilitar desde el GUI. Para habilitarla, debemos abrir un terminal en el host anfitrión y ejecutar los siguientes comandos:

```
VBoxManage modifyvm OVS_01 --nic5 bridged
VBoxManage modifyvm OVS_01 --bridgeadapter5 "ovs1-odl1'
```

5. Con la máquina virtual arrancada abrimos una terminal donde ejecutaremos los comandos para configurar el switch:

   a. Creación del switch:

```
sudo ovs-vsctl add-br br0
```

   b. Añadimos las interfaces creadas al switch (para conocer el nombre de las mismas podemos utilizar el comando "`ifconfig`"):

```
sudo ovs-vsctl add-port br0 enp0s3
sudo ovs-vsctl add-port br0 enp0s8
sudo ovs-vsctl add-port br0 enp0s9
sudo ovs-vsctl add-port br0 enp0s16
```

   c. Comprobamos que se han añadido correctamente:

```
sudo ovs-vsctl show br0
```

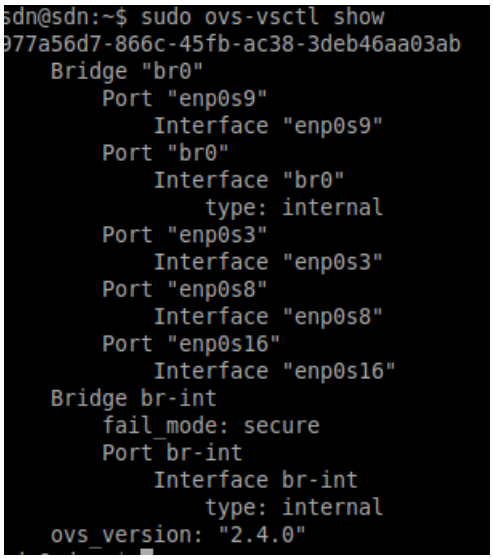


Figura 20: Puertos añadidos a br0

(Fuente: Elaborada para este TFG)

6. Configuración IP: Para conectarnos al controlador debemos añadir una IP del mismo modo que se mostraba en el paso 4 de la configuración de PC_01.

Por último, procedemos a la configuración de la máquina virtual del controlador:

1. Clonamos la máquina virtual.

2. Añadimos una interfaz en modo bridge conectada al adaptador creado anteriormente (ovs1-odl1).

3. En el escritorio, se dispone de un archivo comprimido que contiene la distribución de OpenDaylight. Lo descomprimimos en el directorio que queramos.

4. Abrimos una terminal y nos dirigimos a la raíz del directorio donde hayamos descomprimido el archivo. Una vez aquí arrancamos el controlador:

```
./bin/karaf
```

5. Esperamos hasta que arranque y se nos muestre la siguiente pantalla:

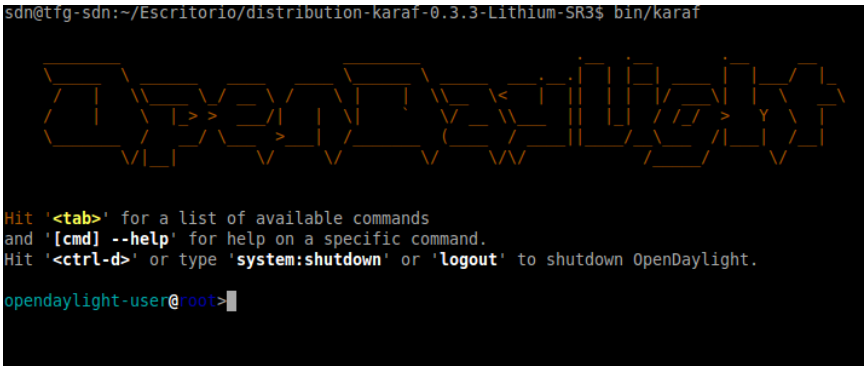


Figura 21: Interfaz de Línea de Comandos de OpenDaylight

(Fuente: Elaborada para este TFG)

Llegados a este punto, solo nos queda conectar nuestro switch al controlador. Para ello, en una terminal dentro de la máquina de OVS_01 ejecutamos el siguiente comando:

```
sudo ovs-vsctl set-controller br0 tcp:200.0.0.1:6633
```

Volvemos a hacer utilizar el comando para visualizar los componentes del switch y vemos que se ha añadido el controlador y se ha conectado correctamente

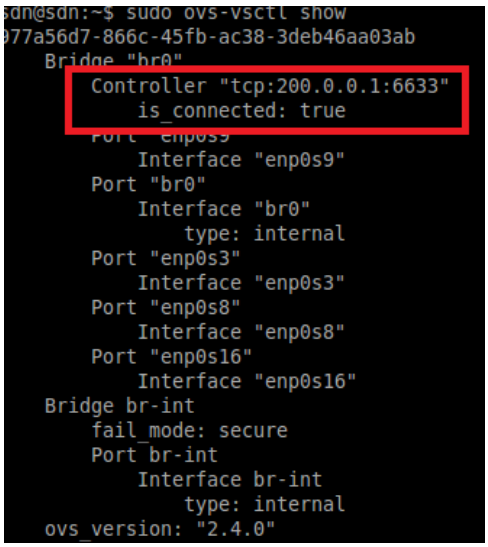


Figura 22: Conexión exitosa entre br0 y controlador

(Fuente: Elaborada para este TFG)

Para ver casos de uso y pruebas con este y otros escenarios puede consultar la sección de pruebas (punto 4 de este documento).

4. ESCENARIOS DE PRUEBAS

En este apartado explicaremos las pruebas realizadas con las diferentes arquitecturas sobre las cuales se ha cimentado el guion de prácticas elaborado.

4.1 LABORATORIO 1: ENTRADAS DE FLUJO OPENFLOW EN OPENVSWITCH

En este primer laboratorio dispondremos de una arquitectura compuesta por tres equipos (máquinas virtuales PC_XX) y un switch (máquina OVS_XX). Para configurarlas y establecer la red hemos seguido la metodología explicada en el apartado 3.3.2 .

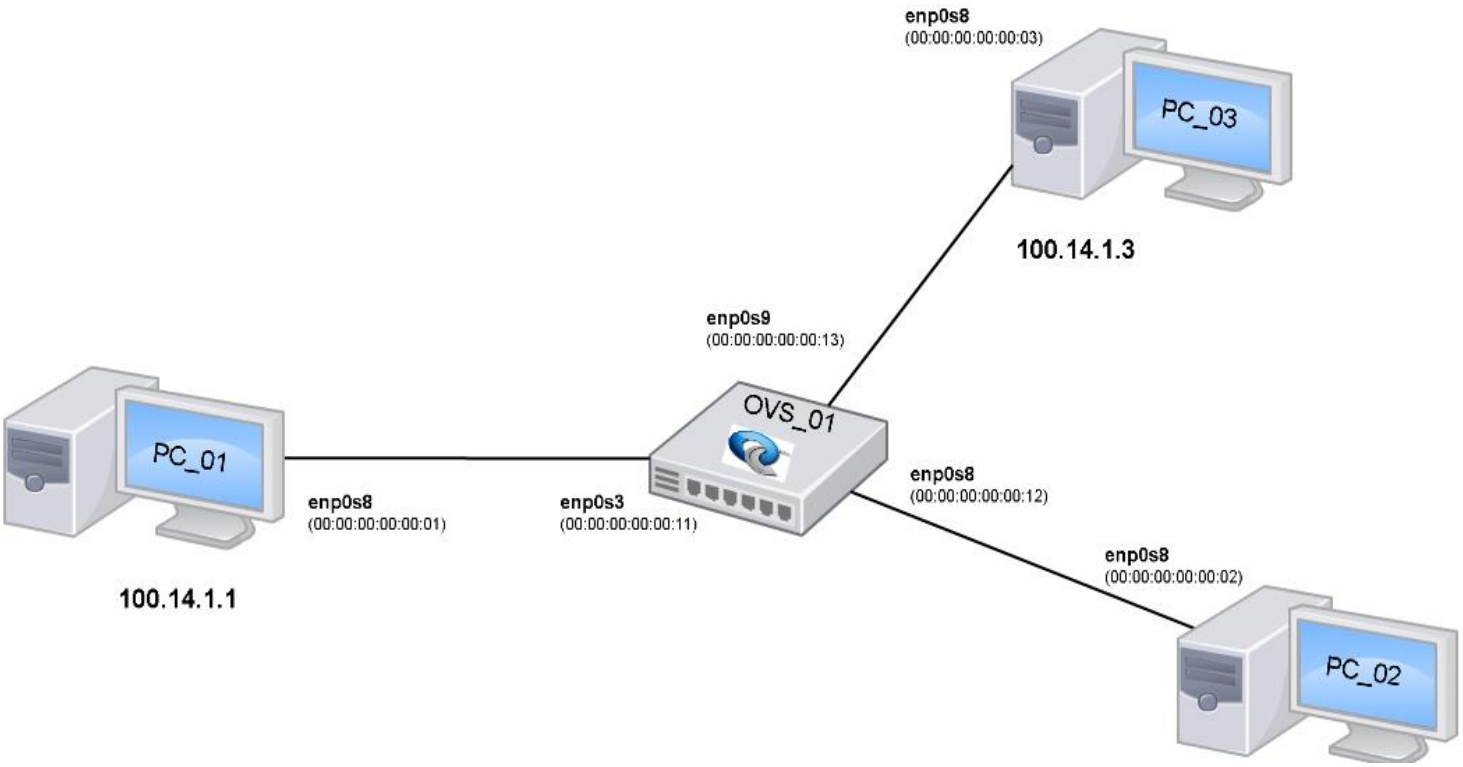


Figura 23: Arquitectura SDN Simple sin Controlador (Laboratorio 1)

(Fuente: Elaborada para este TFG)

4.1.1 PARTE 1 : ROUTING BÁSICO

Una vez tenemos la arquitectura montada, si intentamos hacer ping entre los equipos no obtendremos respuesta ya que la tabla de flujo del switch OpenFlow está vacía.

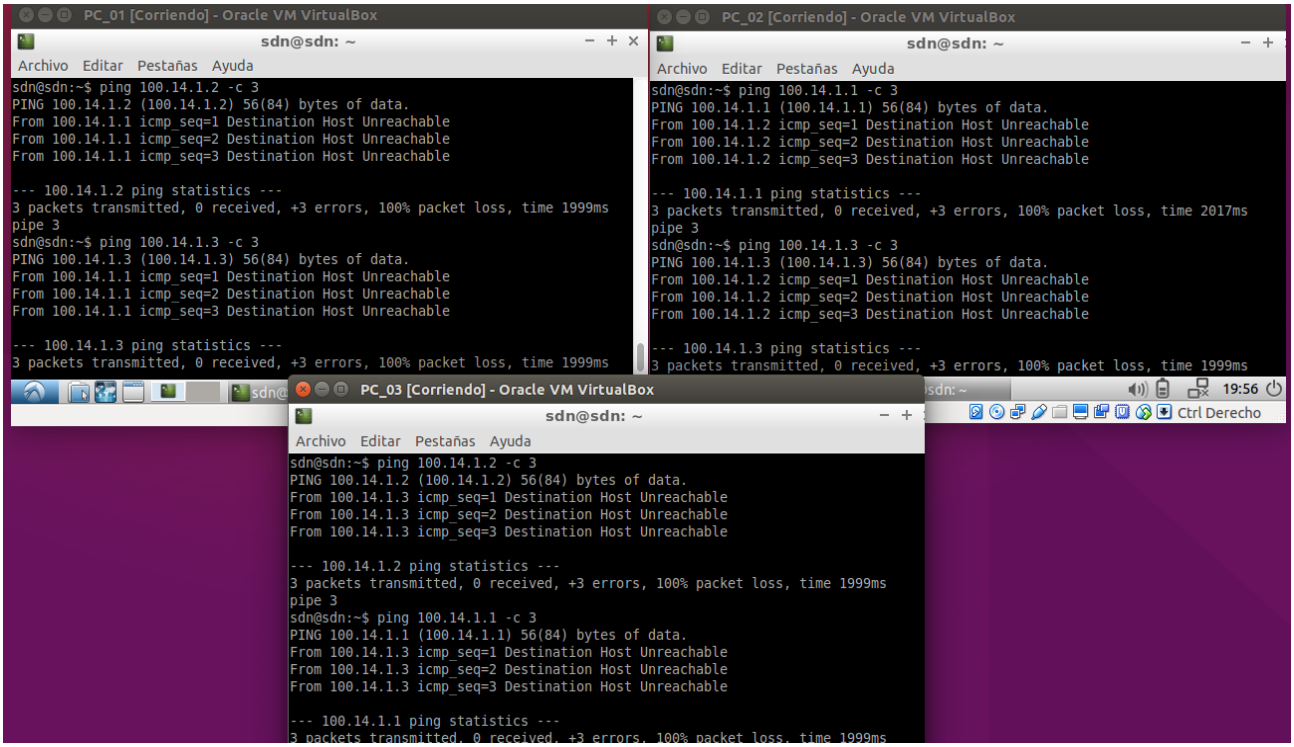


Figura 24: Ping Erróneo entre máquinas (Laboratorio 1, Parte 1)

(Fuente: Elaborada para este TFG)

Para conseguir tener conectividad entre los equipos deberemos añadir flujos a la tabla manualmente simulando la que sería la función de un controlador.

En nuestra primera prueba añadiremos un flujo que le indique al switch que se comporte como un router convencional de capa 2, es decir, que aprenda las rutas de reenvío por sí mismo:

```
sudo ovs-ofctl add-flow br0 action=normal
```

Como resultado, si ejecutamos ahora el ping observaremos que se produce correctamente y veremos cómo en la tabla se ha añadido una nueva entrada.

```
sdn@sdn:~$ sudo ovs-ofctl add-flow br0 action=normal
sdn@sdn:~$ sudo ovs-ofctl dump-flows br0
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=4.045s, table=0, n_packets=0, n_bytes=0, idle_age=4, actions=NORMAL
```

Figura 25: Tabla Flujo br0 (action=normal)

(Fuente: Elaborada para este TFG)

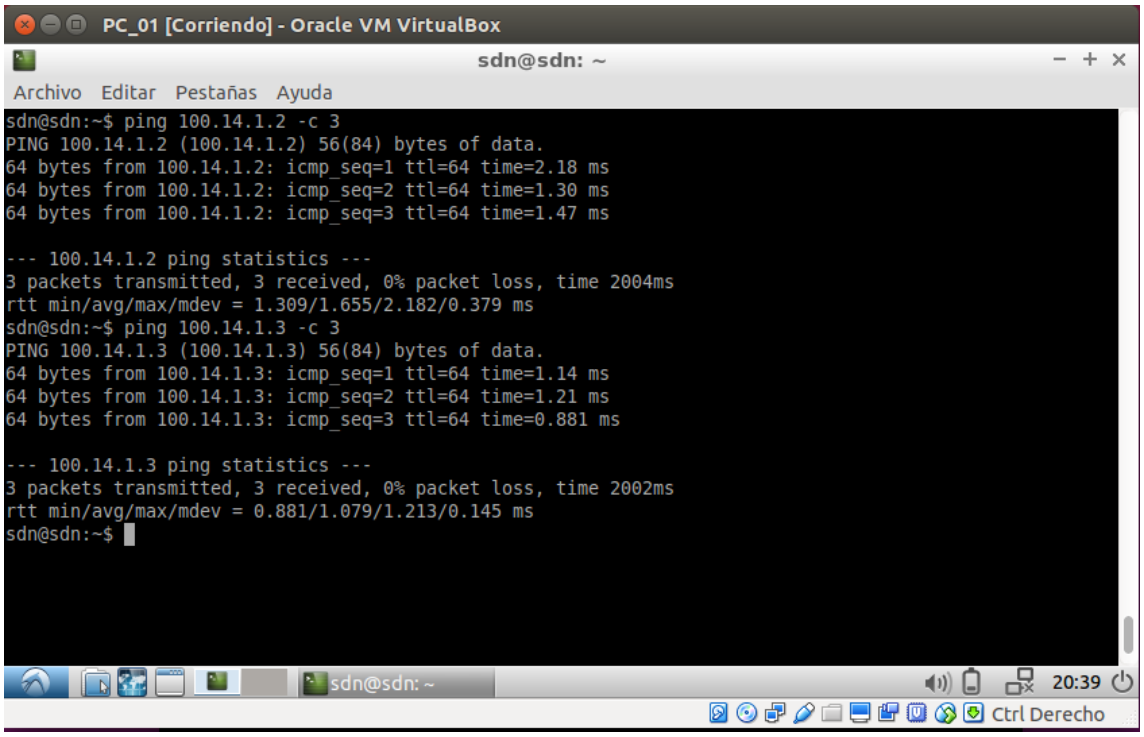


Figura 26: Ping Correcto entre Máquinas (Laboratorio 1, Parte 1)

(Fuente: Elaborada para este TFG)

En la figura 21 podemos comprobar brevemente algunos campos de una entrada OpenFlow:

- **cookie:** Número identificativo (si no se indica al añadir el flujo se pone a 0).
- **duration:** Tiempo en segundos que lleva la entrada de flujo activa en esta tabla.
- **table:** Número de tabla.
- **n_packets:** Número de paquetes que concordaron con esta entrada de flujo.
- **n_bytes:** Número de bytes que coincidieron con esta entrada de flujo.
- **idle_age:** Número de segundos que han pasado desde la última concordancia con esta entrada.
- **actions:** Acciones a realizar sobre los paquetes que concordaron con esta entrada de flujo.

4.1.2 PARTE 2 : CONCORDANCIA DE NIVEL 2

En este apartado probaremos introduciendo nuevos flujos algo más específicos. Por ello partiendo de la arquitectura anterior borramos los flujos del switch y añadimos los nuevos:

```
sudo ovs-ofctl del-flows br0
sudo ovs-ofctl add-flow br0 priority=500,in_port=1,actions=output:2
sudo ovs-ofctl add-flow br0 priority=500,in_port=2,actions=output:1
```

Estos, indican al switch que todo lo que entre por el puerto 2 (correspondiente al PC_01) lo reenvíe por el puerto 1 (correspondiente al PC_02) y viceversa.
Para saber qué puerto corresponde a cada conexión podemos usar el comando mostrado a continuación:

```
sudo ovs-ofctl show br0
```

Comprobamos por tanto, que como se indicaba, el puerto 1 (enp0s8) tiene por MAC la 00:00:00:00:00:12 correspondiente según la arquitectura mostrada en la figura19 a la conexión con el PC_02. De igual modo, podemos hacerlo para el puerto 2 (enp0s3), cuya MAC corresponde a la conexión con el PC_01.

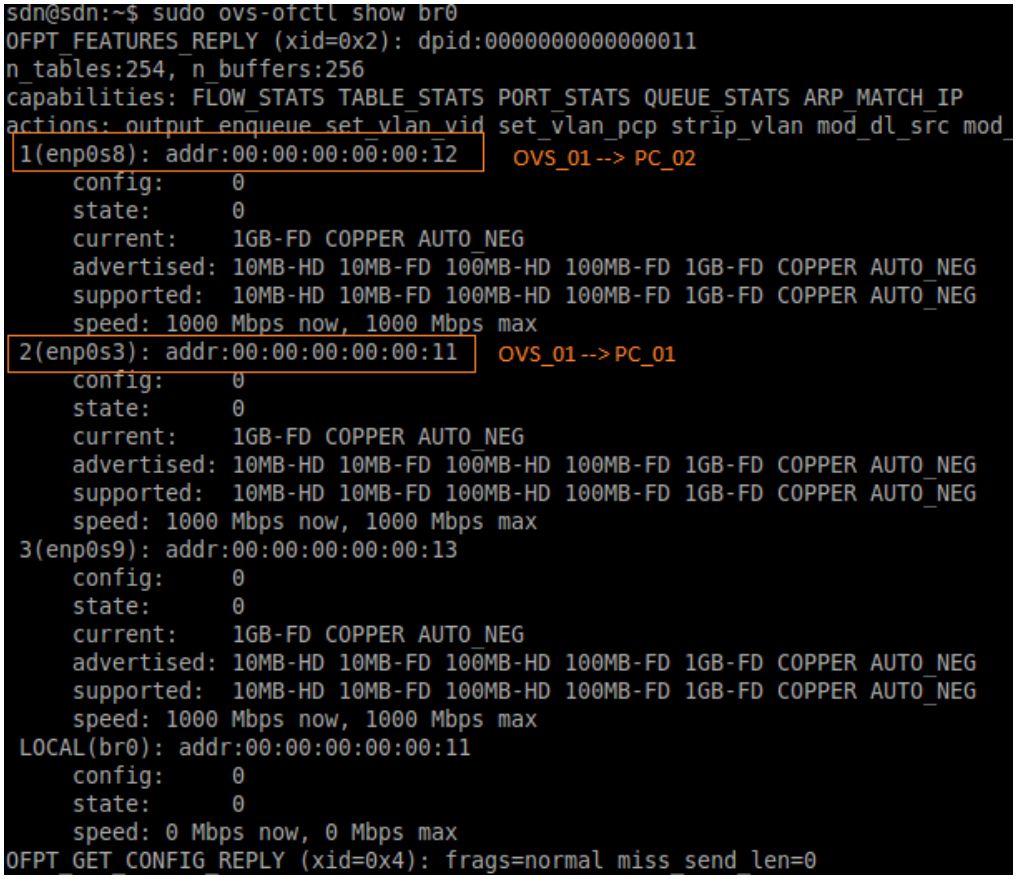


Figura 27: Relación de puertos e interfaces de br0 (Laboratorio 1, Parte 2)

(Fuente: Elaborada para este TFG)

Una vez añadido el flujo, podemos ver como hay conectividad entre los PCs 01 y 02. Sin embargo, si intentamos hacer un ping al PC_03 desde cualquiera de las domáquinas no tenemos conectividad.

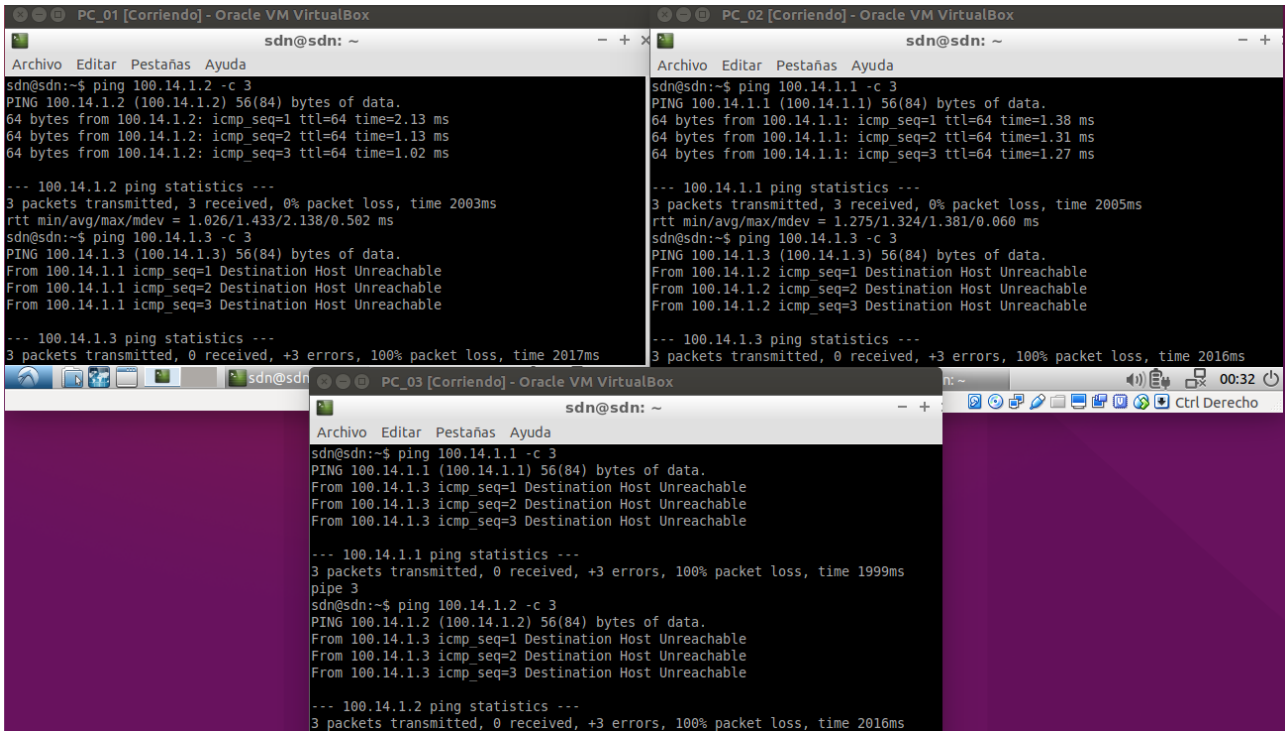


Figura 28: Conectividad tras añadir el primer flujo (Laboratorio 1, Parte 2)

(Fuente: Elaborada para este TFG)

Finalmente, y manteniendo los flujos añadidos, probamos introduciendo una tercera entrada con una prioridad mayor indicando que se descarten todos los paquetes que lleguen al switch:

```
sudo ovs-ofctl add-flow br0 priority=32768,actions=drop
```

De este modo, al llegar un paquete para el que habrá varias coincidencias se realizará la acción correspondiente a la entrada con mayor prioridad. Esto lo podemos comprobar fácilmente haciendo uso del comando mencionado en puntos anteriores para visualizar los flujos de un switch. En la figura 25, observamos como los paquetes se procesan siguiendo la acción marcada por el flujo de mayor prioridad.

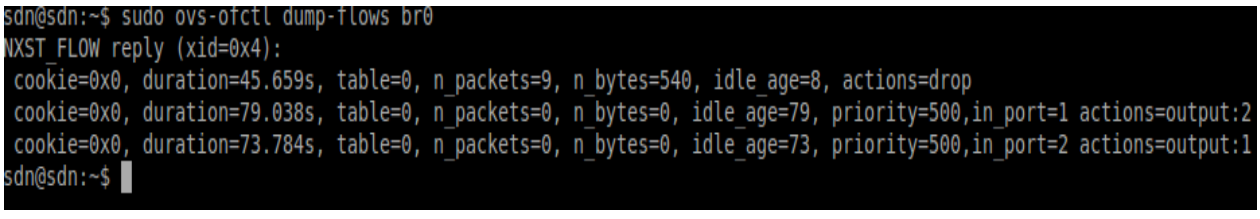


Figura 29: Tabla con entradas de Flujo de diferentes Prioridades (Laboratorio 1, Parte 2)

(Fuente: Elaborada para este TFG)

Tras haber conseguido con éxito enrutar paquetes de un puerto a otro, vamos a proceder a introducir en nuestro switch una entrada diferente, pero que busca un comportamiento muy similar. Le indicaremos que reenvíe los paquetes con MAC destino el PC_01 y origen la del PC_02 por el puerto correspondiente en cada caso, y viceversa:

```
sudo ovs-ofctl add-flow br0
dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,actions=output:1

sudo ovs-ofctl add-flow br0
dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01,actions=output:2
```

Según lo aprendido hasta ahora, deberíamos por lo tanto tener ping. Sin embargo, esto no es posible debido a que, cuando PC_01 quiere contactar con la IP correspondiente al PC_02, lo primero que realiza (al no tener en su tabla ARP una MAC asociada para esa IP) es un ARP Request que envía con dirección MAC de destino la de broadcast (FF: FF:FF:FF:FF:FF) de manera que al llegar al switch y no existir ninguna entrada de flujo que corresponda con él, se descarta el paquete.

```
00:00:00_00:00:01    Broadcast           ARP          60 Who has 100.14.1.2?  Tell 100.14.1.1
00:00:00_00:00:01    Broadcast           ARP          60 Who has 100.14.1.2?  Tell 100.14.1.1
00:00:00_00:00:01    Broadcast           ARP          60 Who has 100.14.1.2?  Tell 100.14.1.1
```

Figura 30: ARP Request emitido por PC_01 (Laboratorio 1, Parte 2)

(Fuente: Elaborada para este TFG)

Para tener conexión correctamente, debemos por tanto añadir un nuevo flujo:

```
sudo ovs-ofctl add-flow br0 dl_type=0x806,nw_proto=1,actions=flood
```

Este indicará que para todo paquete con código de tipo Ethernet 0x806 (el utilizado para el protocolo ARP) y Opcode 1 (correspondiente a los mensajes de tipo request en ARP) se reenvíe el paquete por todos los puertos del switch excepto por el que se recibió (flood).

Si realizamos ping ahora comprobaremos que hay conectividad sin problemas entre las máquinas 1 y 2 pero como en el ejemplo anterior sigue sin haberlo con la máquina 3.

4.1.3 PARTE 3 : CONCORDANCIA A NIVEL IP

Ya hemos comprobado que nuestro switch OpenFlow responde positivamente en situaciones de concordancia de puertos o de direcciones físicas. A continuación, se pretende dar un paso más y probar añadiendo flujos que permitan matcheos a nivel IP. Para ello, añadimos un simple comando que filtra por tipo de protocolo IP (0x800) y que permite añadir una lógica dentro de la subred.

```
sudo ovs-ofctl add-flow br0
priority=500,dl_type=0x800,nw_src=100.14.1.0/24,nw_dst=100.14.1.0/24,a
ctions=normal
```

Una vez más, y como sucedía en la situación anterior debemos añadir una entrada que permita enrutar las peticiones ARP dirigidas a una IP en concreto. Dado el amplio

abanico de posibilidades que ofrece la programabilidad SDN, probaremos con otro comando diferente al utilizado anteriormente:

```
sudo ovs-ofctl add-flow br0 arp,nw_dst=100.14.1.1,actions=output:2
sudo ovs-ofctl add-flow br0 arp,nw_dst=100.14.1.2,actions=output:1
sudo ovs-ofctl add-flow br0 arp,nw_dst=100.14.1.3,actions=output:3
```

De este modo indicamos a nuestro switch que reenvíe los paquetes ARP (sean Request o Reply) dirigidos a la IP que se solicite, directamente al puerto correspondiente, permitiéndonos tener ping entre todas las máquinas.

Aquí podemos ver una de las claras ventajas de tener una visión global de la red que controlamos, ya que no se congestionan los enlaces de manera innecesaria. A modo de ejemplo, si PC_01 quisiera saber la MAC de PC_02, al llegar el ARP Request al switch directamente lo reenviaría por el puerto correcto sin congestionar el enlace con PC_03. Esto en un escenario como el manejado no es especialmente relevante, ya que hay pocos equipos y poco flujo de tráfico de datos, pero en un escenario con miles de equipos y una tasa de flujo mucho mayor se convierte en un factor determinante.

Otra de las ventajas de poder utilizar entradas OpenFlow es que nos permite diferenciar tráficos de forma fácil y sencilla. Como veíamos en el punto 3 de este documento, muchas de las empresas que empezaban a implementar soluciones basadas en SDN utilizaban dicha función para diferenciar los tráficos experimentales y los de producción. En nuestra prueba trataremos de emular dicho comportamiento diferenciando el tráfico proveniente del PC_03.

Para ello, añadiremos una entrada que diferencie el tipo de servicio modificando el valor del campo DSCP en los paquetes con origen PC_03. Le daremos un valor (elegido de manera arbitraria) de 46.

De modo que, si:

$$ToS = DSCP_{(6\ bits)} + ENC_{(2\ bits)}$$

Y queremos un valor de DSCP = 46 (101110 en binario):

$$ToS = 10111000_{bin} = 184_{dec}$$

Y en consecuencia, el flujo a añadir deberá modificar dicho campo por un valor de 184 (observar que es preciso dar una prioridad mayor a este flujo respecto a los anteriores para obtener el comportamiento deseado):

```
sudo ovs-ofctl add-flow br0
priority=800,ip,nw_src=100.14.1.3,actions=mod_nw_tos:184,normal
```

En las figuras 27 y 28 podemos ver una comparativa entre los paquetes recibidos desde el PC_03 antes y después de añadir nuestra entrada de flujo.

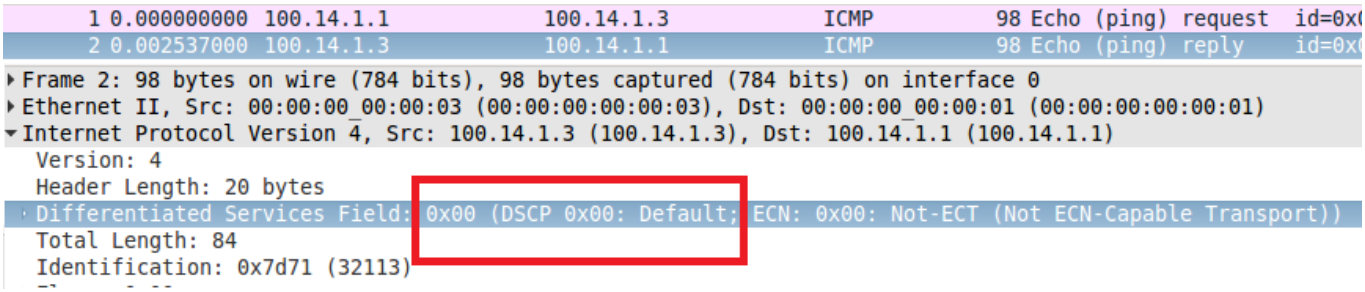


Figura 27: Captura paquete PC_03 antes de cambiar DSCP (Laboratorio 1, Parte 3)

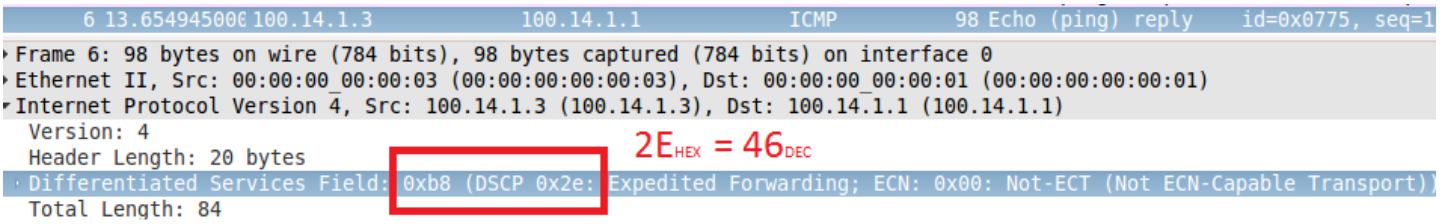


Figura 31: Captura Paquete PC_03 después de introducir entrada de flujo para cambiar DSCP (Laboratorio 1, Parte 3)

(Fuente: Elaborada para este TFG)

4.1.4 PARTE 4 : IMPLEMENTACIÓN DE UN FIREWALL TCP

Por último, para terminar nuestras pruebas implementaremos un sencillo firewall TCP, de manera que el PC_03 hará las funciones de servidor y el switch OpenFlow hará las veces de cortafuegos impidiendo que pasen los paquetes que no vayan dirigidos al puerto 80.

Para ello, partiendo de una tabla de flujo vacía, añadimos unas sencillas entradas que aseguraran el correcto funcionamiento de la red entre los clientes y el servidor:

```
sudo ovs-ofctl add-flow br0 arp,actions=normal

sudo ovs-ofctl add-flow br0
priority=800,nw_src=100.14.1.3,actions=normal
```

Finalmente, introducimos el flujo que indicará que los paquetes TCP (nw_proto=6) con destino el puerto 80 se encaminen por el puerto 3 del switch:

```
sudo ovs-ofctl add-flow br0
priority=500,dl_type=0x800,nw_proto=6,tp_dst=80,actions=output:3
```

Fácilmente podemos comprobar como únicamente llegan al PC_03 los paquetes de tipo TCP encaminados al puerto 80 haciendo uso de la herramienta iperf. Lanzamos dos servidores a la vez en el PC_03 (uno escuchando por el puerto 80 y otro por el 90).Vemos como si intentamos contactar por el puerto 80 se establece la conexión, mientras que si lo hacemos desde el puerto 90 no se recibe ningún paquete, ya que como indicábamos, el switch realizará las funciones de firewall.

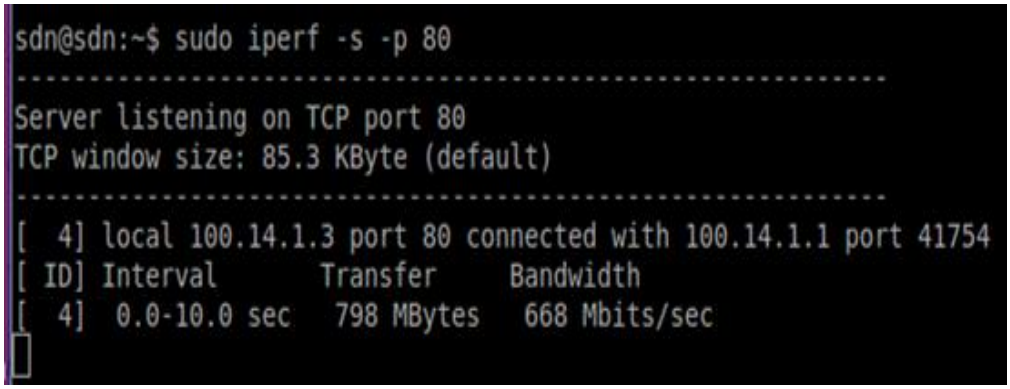


Figura 32 Conexión TCP establecida entre PC_01 y PC_02 (Laboratorio 1, Parte 4)

(Fuente: Elaborada para este TFG)

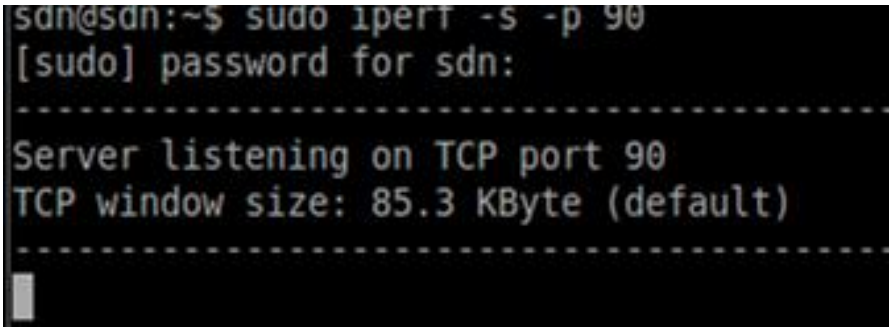


Figura 33: Conexión TCP no establecida (Laboratorio 1, Parte 4)

(Fuente: Elaborada para este TFG)

4.2 LABORATORIO 2 : OPERACIONES BÁSICAS CON OPENDAYLIGHT

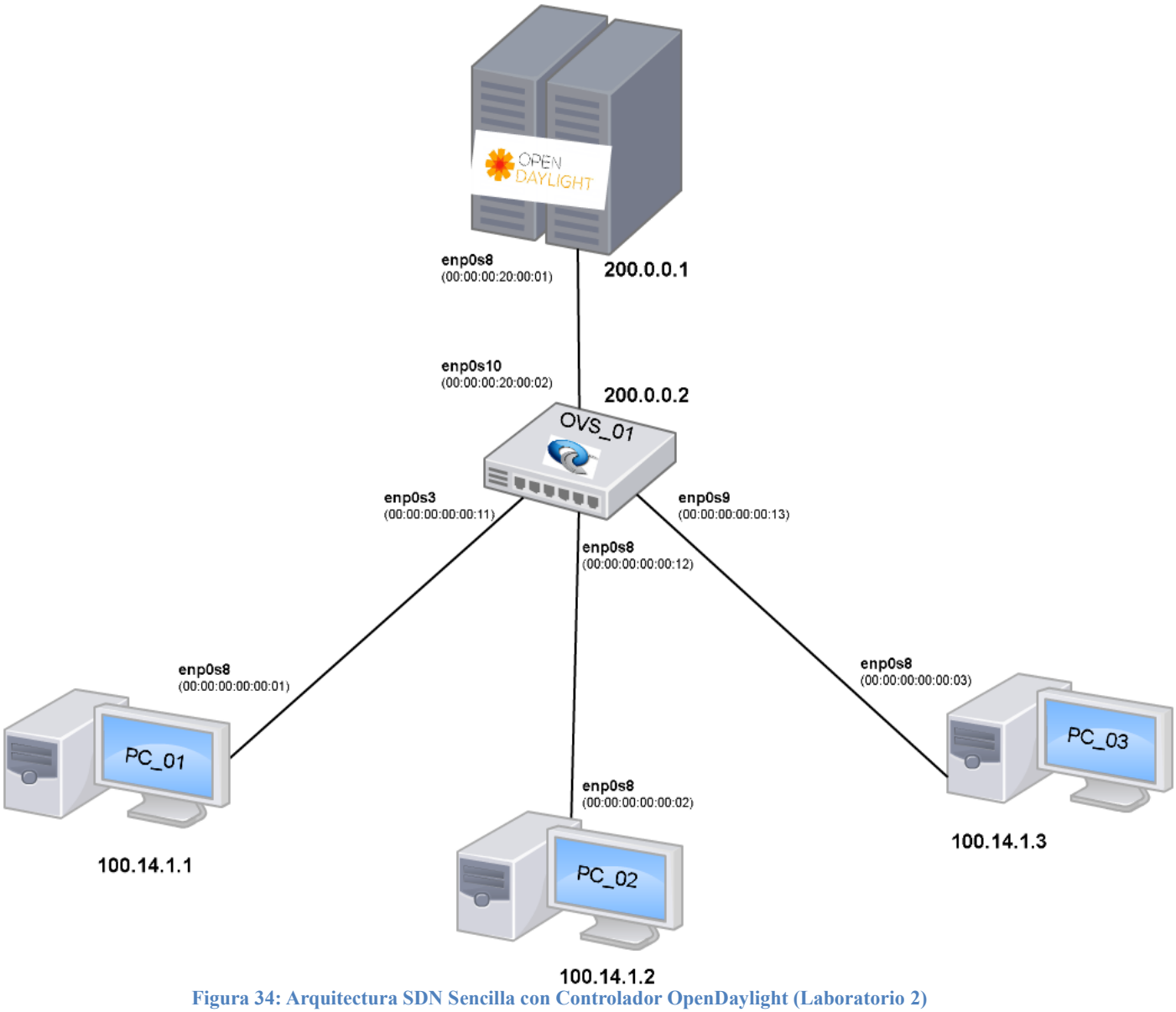


Figura 34: Arquitectura SDN Sencilla con Controlador OpenDaylight (Laboratorio 2)

(Fuente: Elaborada para este TFG)

Una vez que hemos adquirido cierta soltura con las pruebas realizadas en el apartado anterior, sobre cómo gestiona un switch OpenFlow sus los flujos y cómo los utiliza, en este segundo laboratorio utilizaremos la misma arquitectura que en el laboratorio 1 añadiendo una máquina virtual que actuará como controlador SDN como se refleja en la figura 31. Cómo comentábamos en el apartado 3.2.3 sobre las decisiones de diseño utilizaremos OpenDaylight.

Para el despliegue de la arquitectura seguimos los pasos indicados en el punto 3.3.2 de este documento, de manera que tengamos el controlador arrancado y el switch correctamente conectado.

El objetivo planteado es familiarizarnos con el entorno de OpenDaylight (tanto en su interfaz de línea de comandos como en su interfaz gráfica web) y entender los métodos que podemos utilizar para sacarle provecho.

4.2.1 INSTALACIÓN DE UNA FEATURE

Lo primero que aprenderemos para familiarizarnos con Karaf (interfaz de usuario por línea de comandos utilizada por OpenDaylight), es a instalar una de las utilidades disponibles en nuestra distribución de ODL. Está feature viene incluida para ser instalada sin necesidad de conexión a internet.

Añadiremos una feature que permitirá al controlador procesar los paquetes bajo la lógica de un switch de capa 2.

Supongamos que no sabemos el nombre exacto de dicha utilidad. Si hacemos uso del comando mostrado a continuación, nos imprimirá por pantalla una lista de todas las features disponibles:

```
feature:list
```

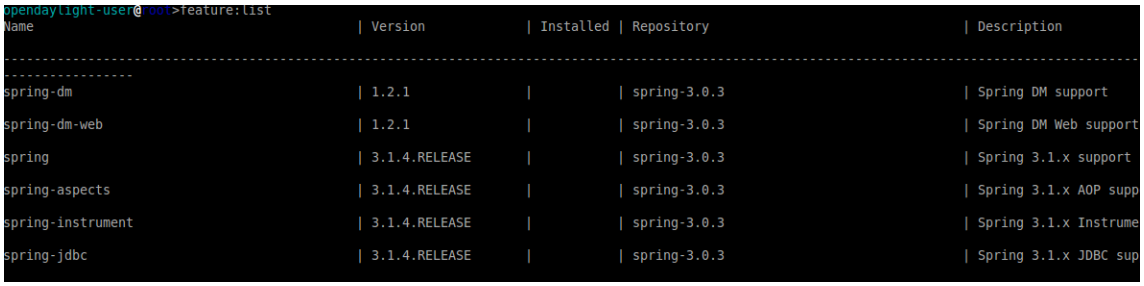


Figura 35: Algunas de las features disponibles para instalar (Laboratorio 2)

(Fuente: Elaborada para este TFG)

La funcionalidad que añadiremos se llama "odl-l2switch-switch-ui". Si queremos saber más acerca de nuestra feature y qué bundles y/o features se instalarán de forma adicional para su correcto funcionamiento podemos hacerlo con el comando `"feature:info odl-l2switch-switch-ui"`

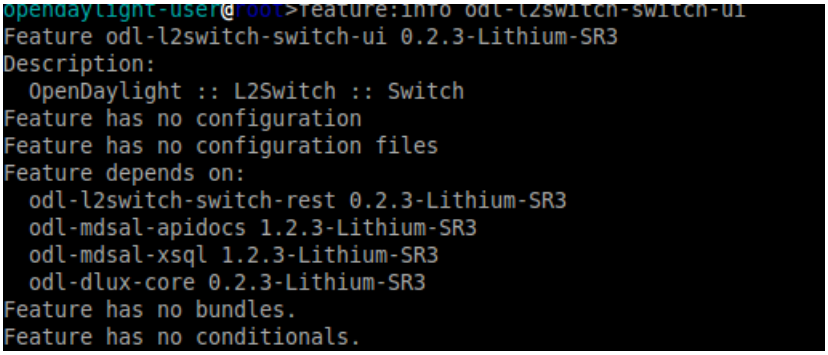


Figura 36: Información de la Feature odl-l2switch-switch-ui (Laboratorio 2)

(Fuente: Elaborada para este TFG)

Los últimos dos comandos nos ayudan a entender cómo utilizar Karaf en caso de no saber nada acerca de las aplicaciones disponibles, pero para realmente instalar una feature deberemos hacerlo ejecutando el comando `"feature:install odl-l2switch-switch-ui"`. Otra forma de instalar una aplicación es modificando el archivo de configuración antes de arrancar el controlador. Este se encuentra en `"<carpeta distribución>/etc/org.apache.karaf.features.cfg"` . Para añadir una nueva feature a la lista, simplemente añadimos su nombre a la línea `"featuresBoot=<feature1>, <feature2>,…<featureN>"`.

Podemos comprobar que se ha instalado correctamente usando el comando `"bundle:list -s | grep l2switch"`.

```
opendaylight-user@root>bundle:list -s | grep l2switch
200 | Active   |  80 | 0.2.3.Lithium-SR3                | org.opendaylight.l2switch.packethandler.model
201 | Active   |  80 | 0.2.3.Lithium-SR3                | org.opendaylight.l2switch.packethandler.impl
207 | Active   |  80 | 0.2.3.Lithium-SR3                | org.opendaylight.l2switch.loopremover.model
208 | Active   |  80 | 0.2.3.Lithium-SR3                | org.opendaylight.l2switch.loopremover.impl
210 | Active   |  80 | 0.2.3.Lithium-SR3                | org.opendaylight.l2switch.addresstracker.model
211 | Active   |  80 | 0.2.3.Lithium-SR3                | org.opendaylight.l2switch.arphandler.impl
212 | Active   |  80 | 0.2.3.Lithium-SR3                | org.opendaylight.l2switch.addresstracker.impl
213 | Active   |  80 | 0.2.3.Lithium-SR3                | org.opendaylight.l2switch.hosttracker.model
214 | Active   |  80 | 0.2.3.Lithium-SR3                | org.opendaylight.l2switch.hosttracker.impl
215 | Active   |  80 | 0.2.3.Lithium-SR3                | org.opendaylight.l2switch.main.impl
opendaylight-user@root>
```

Figura 37: Lista de aplicaciones instaladas filtrando por nombre (Laboratorio 2)

(Fuente: Elaborada para este TFG)

4.2.2 LA INTERFAZ GRÁFICA DE USUARIO DE OPENDAYLIGHT: DLUX

Una vez hemos aprendido cómo instalar una feature, vamos a descubrir las ventajas de trabajar con la interfaz gráfica web de OpenDaylight.

En primer lugar, en la consola de Karaf instalamos todas las features referentes a Dlux (GUI de ODL) con el comando `"feature:install odl-dlux-all"`

Para acceder a la interfaz gráfica de usuario proporcionada por ODL debemos abrir un navegador en la máquina virtual y dirigirnos a la dirección http://<IP_controlador>:8181/index.html . Si lo hacemos desde la propia máquina virtual del controlador, podemos acceder usando la palabra clave "localhost" en lugar de la IP del controlador. Nos aparecerá una pantalla para iniciar sesión donde los credenciales son "admin/admin".

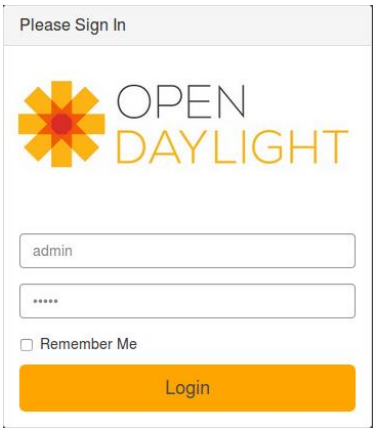


Figura 38: Pantalla de Inicio de Sesión de DLUX (Laboratorio 2)

(Fuente: Elaborada para este TFG)

Tras iniciar sesión nos debería aparecer una imagen con la arquitectura que el controlador puede ver aunque es posible que aparezca vacía. Esto es debido a que el controlador tarda en arrancar todos sus módulos.

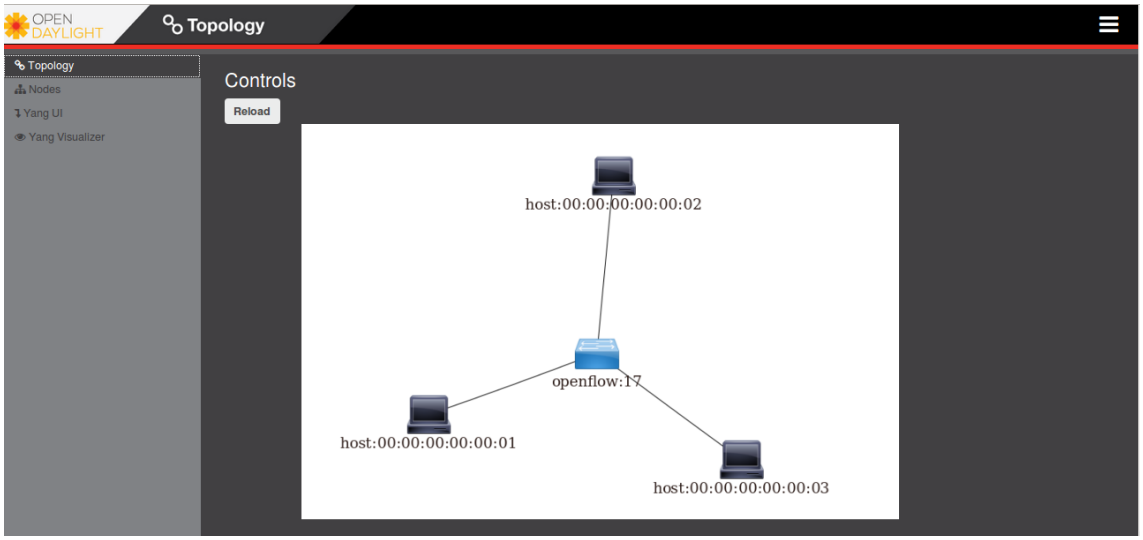


Figura 39: Arquitectura vista por el Controlador (Laboratorio 2)

(Fuente: Elaborada para este TFG)

Para asegurarnos, podemos observar los mensajes de carga de módulos del LOG, donde se indica cuando está 100% operativo.

```
6R3 | Successfully pushed configuration snapshot 10-rest-connector.xml(odl-dlux-core,odl-dlux-core)
2016-06-18 18:10:56,242 | INFO  | config-pusher     | ConfigPusherImpl                  | 135 - org.opendaylig
6R3 | Pushing configuration snapshot 10-restconf-service.xml(odl-dlux-core,odl-dlux-core)
2016-06-18 18:10:56,443 | INFO  | config-pusher     | ConfigPusherImpl                  | 135 - org.opendaylig
6R3 | Successfully pushed configuration snapshot 10-restconf-service.xml(odl-dlux-core,odl-dlux-core)
2016-06-18 18:10:56,443 | INFO  | config-pusher     | ConfigPusherImpl                  | 135 - org.opendaylig
6R3 | Pushing configuration snapshot 04-xsql.xml(odl-mdsal-xsql,odl-mdsal-xsql)
2016-06-18 18:10:56,670 | INFO  | config-pusher     | ConfigPusherImpl                  | 135 - org.opendaylig
6R3 | Successfully pushed configuration snapshot 04-xsql.xml(odl-mdsal-xsql,odl-mdsal-xsql)
```

Figura 40: Log ODL: Módulos cargados correctamente (Laboratorio 2)

(Fuente: Elaborada para este TFG)

Podemos por tanto comprobar ahora como el propio controlador (siguiendo la lógica de un router de capa 2), automáticamente ha generado unas entradas en la tabla de flujos del switch, que administran la red y permiten conectividad entre todas las máquinas.

Figura 41: Flujos añadidos por el Controlador (Laboratorio 2)

(Fuente: Elaborada para este TFG)

PETICIONES MEDIANTE LA INTERFAZ REST Y HERRAMIENTAS DE DLUX

Para continuar con las pruebas en torno al funcionamiento de la interfaz gráfica, utilizaremos una interfaz REST. Esta, es una interfaz con acceso directo a las configuraciones dentro de nuestro controlador desde el directorio la URL http://localhost:8181/restconf/* . Dentro de este, se diferencian dos sub-rutas:

- **restconf/config/*:** Aquí los usuarios (generalmente administradores de red) pueden crear, actualizar, consultar o eliminar (POST, PUT, GET, DELETE) la configuración de las aplicaciones.
- **restconf/operational/*:** Aquí es dónde las aplicaciones escriben los estados y donde los usuarios pueden consultarlos (GET). Las bases de datos más consultadas dentro de este directorio, son la topología y el inventario de nodos.

Así pues podemos utilizar diferentes herramientas para acceder a estos directorios. Para este laboratorio vamos a utilizar en un primer lugar la herramienta disponible en Dlux YangUI (herramienta que permite buscar URLs y métodos del controlador). Para ello, pinchamos en YangUI → opendaylight-inventory →operational→nodes. En el desplegable de abajo seleccionamos GET como método y presionamos en enviar. Este request nos devolverá un esquema con todos los nodos de la red.

Como resultado, podenos comprobar en la figura 38 como tenemos una lista con todos los switches (“node list”) y los equipos o interfaces conectados (“node-connector list”) . Aquí podemos consultar los paquetes enviados por cada equipo, sus IPs, direcciones MAC,…

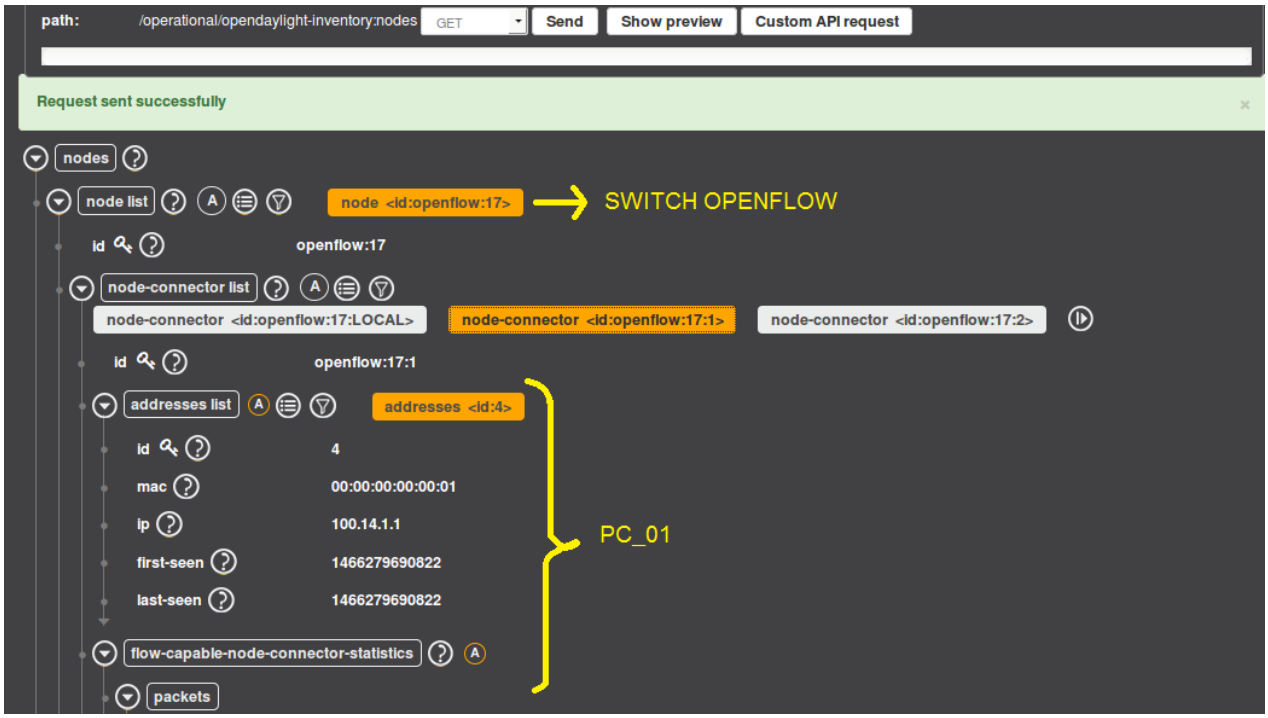


Figura 42: Topología vista en YangUI (Laboratorio 2)

(Fuente: Elaborada para este TFG)

De manera equivalente podemos consultar la topología usando el comando `"curl"` de Linux con el siguiente formato:

```
curl --user <user>:<password> -H <cabecera1> … -H <cabeceraN> -X<tipo de petición http> <url> -d '<cuerpo de la petición>'
```

Para nuestro caso la petición sería:

```
curl --user "admin":"admin" -H "Accept:application/xml" -H "Content-type:application/xml" -X GET http://localhost:8181/operational/network-topology:network-topology
```

```
-<node>
  <node-id>host:00:00:00:00:00:01</node-id>
 -<termination-point>
    <tp-id>host:00:00:00:00:00:01</tp-id>
  </termination-point>
  <id>00:00:00:00:00:01</id>
 -<attachment-points>
    <tp-id>openflow:17:1</tp-id>
    <active>true</active>
    <corresponding-tp>host:00:00:00:00:00:01</corresponding-tp>
  </attachment-points>
 -<addresses>
    <id>4</id>
    <ip>100.14.1.1</ip>
    <last-seen>1466279690822</last-seen>
    <mac>00:00:00:00:00:01</mac>
    <first-seen>1466279690822</first-seen>
  </addresses>
</node>
```

Figura 43: XML PC_01 (Laboratorio 2)

(Fuente: Elaborada para este TFG)

4.3 LABORATORIO 3 : BALANCEADOR DE CARGA COMO SERVICIO

En este apartado emplearemos una arquitectura prácticamente idéntica a la mostrada en el apartado anterior añadiendo un host más que hará de cliente (PC_04).

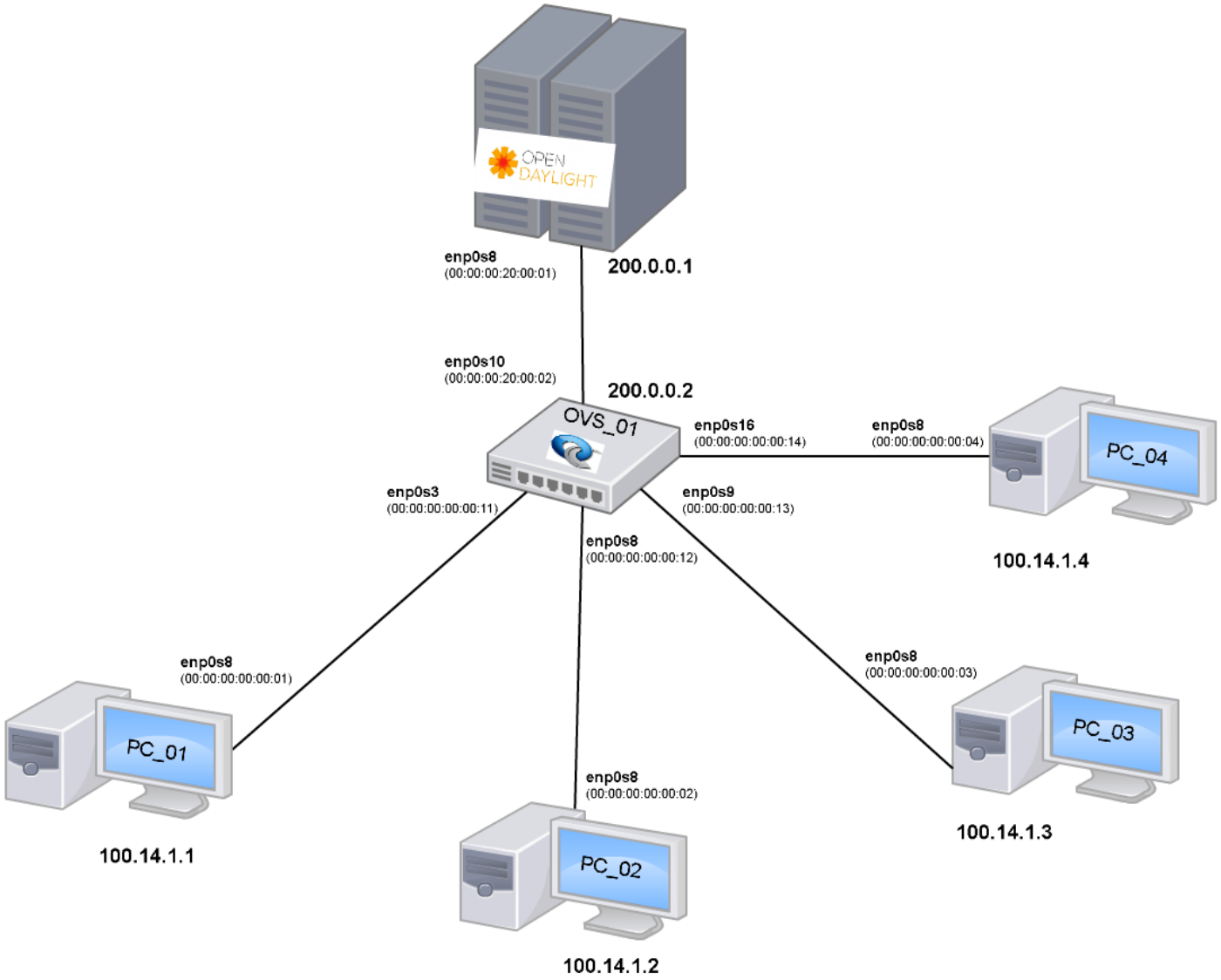


Figura 44: Arquitectura SDN Balanceador de carga Sencillo (Laboratorio 3)

(Fuente: Elaborada para este TFG)

Las pruebas realizadas en este apartado pretenden emular el funcionamiento de un balanceador de carga que siga un algoritmo de Round Robin para alternar las peticiones a sus servidores (PC_01, PC_02 y PC_03). Para ello utilizaremos el servicio proporcionado por OpenDaylight Balanceo de carga como Servicio (LbasS, Load Balancing as a Service) que nos permitirá configurar dicho servicio utilizando una REST API. Para comunicarnos con el controlador utilizaremos la herramienta "curl", la cual ya vimos en funcionamiento en el apartado anterior.

En primer lugar crearemos un pool en nuestro controlador, donde se indicarán los equipos que actuarán como servidores:

```
curl -- user "admin":"admin" -H "Accept:application/json" -H "Content-type:application/json" -X POST http://localhost:8080/one/nb/v2/lb/default/create/pool -d '{"name":"PoolRR","lbmethod":"roundrobin"}'
```

A continuación, creamos la IP virtual (VIP) a la que deberán ir dirigidas las peticiones, indicando protocolo y puerto:

```
curl -- user "admin":"admin" -H "Accept:application/json" -H "Content-type:application/json" -X POST http://localhost:8080/one/nb/v2/lb/default/create/vip -d '{"name":"VIP-RR","ip":"100.14.1.20","protocol":"TCP","port":"5550","poolname":"PoolRR"}'
```

Añadimos los miembros del pool que harán de servidores:

```
curl -- user "admin":"admin" -H "Accept:application/json" -H "Content-type:application/json" -X PUT http://localhost:8080/one/nb/v2/lb/default/create/poolmember -d '{"name":"Server1","ip":"100.14.1.1",poolname":"PoolRR"}'

curl -- user "admin":"admin" -H "Accept:application/json" -H "Content-type:application/json" -X PUT http://localhost:8080/one/nb/v2/lb/default/create/poolmember -d '{"name":"Server2","ip":"100.14.1.2",poolname":"PoolRR"}'

curl -- user "admin":"admin" -H "Accept:application/json" -H "Content-type:application/json" -X PUT http://localhost:8080/one/nb/v2/lb/default/create/poolmember -d '{"name":"Server3","ip":"100.14.1.3",poolname":"PoolRR"}'
```

Por último, es necesario añadir una entrada ARP de manera manual a la tabla de flujo del cliente (PC_04). Esta entrada es necesaria, ya que la IP virtual creada anteriormente no corresponde a ningún equipo. En consecuencia, los mensajes ARP Request desde el cliente no obtendrían nunca una respuesta y no se enviarían las peticiones:

```
sudo arp -s 100.14.1.20 00:00:00:00:00:20
```

Utilizamos la herramienta iperf para simular el funcionamiento de cliente y servidores TCP:

```
iperf -s -p 5550 (en los servidores)
iperf -c 100.14.1.20 -p 5550 (en el cliente)
```

Una vez arrancados, al solicitar una conexión TCP desde el cliente deberíamos comprobar cómo se conecta cada vez a un servidor siguiendo el algoritmo marcado (PC_01→ PC_02→ PC_03 y vuelta a empezar).

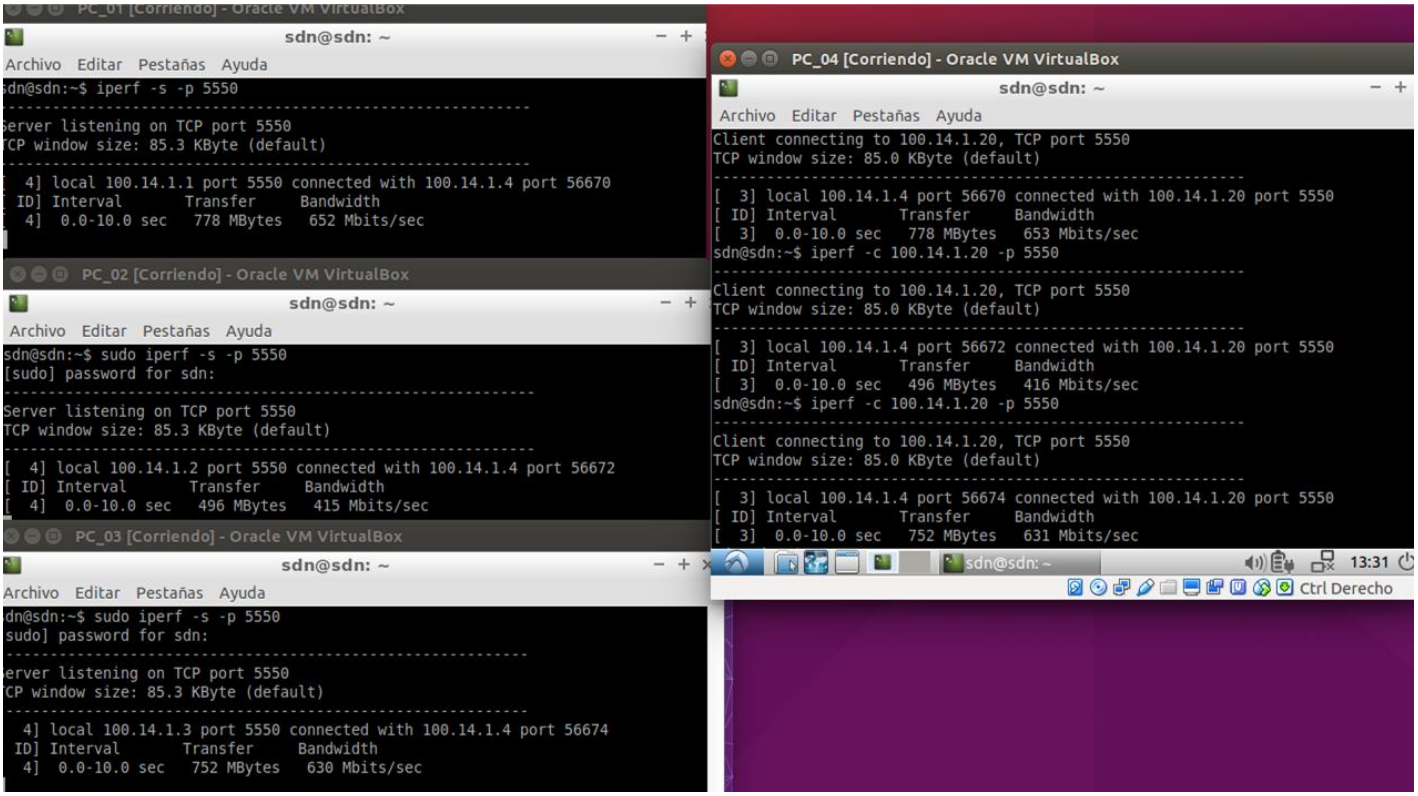


Figura 45: Balanceo de Carga siguiendo un algoritmo Round-Robin (Laboratorio 3)

(Fuente: Elaborada para este TFG)

Si lanzamos wireshark podemos observar como el proceso es transparente para el cliente. Así pues, cada vez que se recibe una petición de conexión con el servidor (en la IP 100.14.1.20) el switch delega en el controlador (mediante el envío de un PACKET_IN) y este decide qué hacer con el flujo (decide a qué servidor le toca atender la petición).

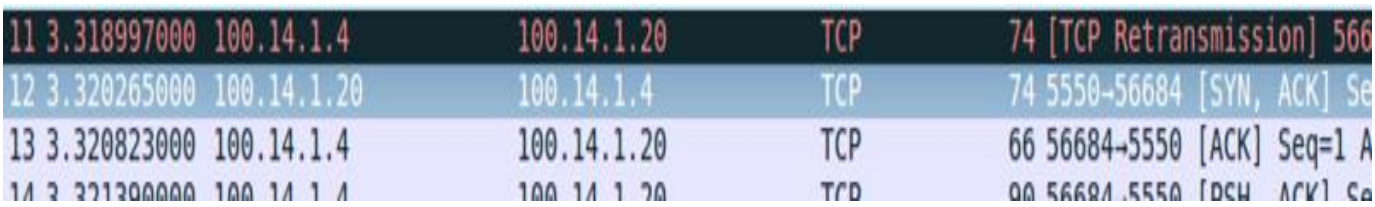


Figura 46: Conexión desde el punto de vista del PC_04 (Laboratorio 3)

(Fuente: Elaborada para este TFG)

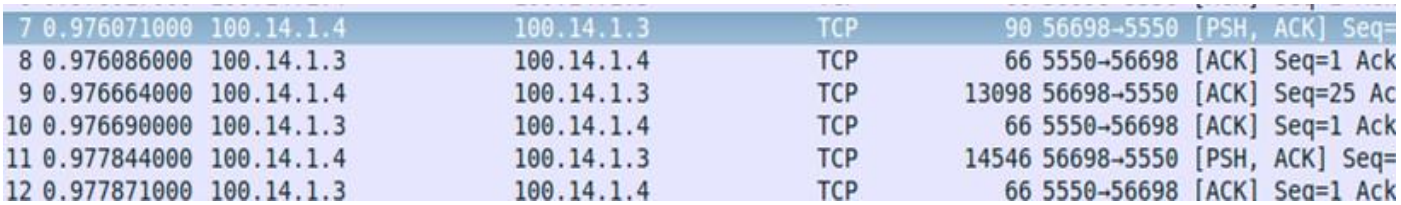


Figura 48: Conexión desde el punto de vista del Servidor 3 (Laboratorio 3)

(Fuente: Elaborada para este TFG)

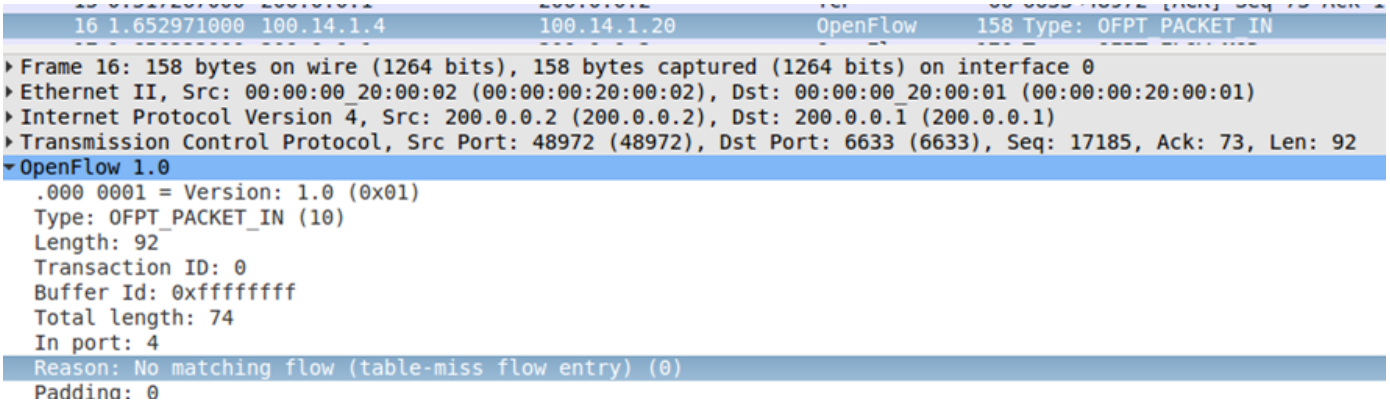


Figura 47: Packet_In recibido en el Controlador (Laboratorio 3)

(Fuente: Elaborada para este TFG)

4.4 LABORATORIO 4 : INSTALACIÓN DE UNA APLICACIÓN JAVA EN NUESTRO CONTROLADOR

En este apartado se tiene como objetivo aprender a instalar una aplicación propia. Dado lo laborioso del asunto y de que no es el objetivo principal de este trabajo utilizaremos una sencilla aplicación desarrollada por el grupo SdnHub y que al igual que hacía la feature "odl-l2-switch" del punto 4.2 de este documento, añadirá una lógica a nuestro controlador.

Previamente a montar la arquitectura (la cual es equivalente a la del laboratorio 2 sustituyendo la máquina del controlador por la de SdnHub), debemos montar el proyecto utilizando Maven (herramienta de software para la gestión y construcción de proyectos). Para ello, iniciamos la máquina "SdnHub", en la que disponemos de una versión instalada de OpenDaylight para desarrolladores.

En primer lugar, lo que debemos hacer es montar el proyecto usando el siguiente comando en la ruta donde se encuentra instalado OpenDaylight:

```
mvn clean install -DskipTest -nsu
```

Si el proceso va bien, veremos en la pantalla un mensaje "BUILD SUCCESS" que nos indicará que se ha realizado el montaje con éxito.

```
-SNAPSHOT/packaging-1.0.0-SNAPSHOT.pom
[INFO] Installing /home/ubuntu/SDNHub_Opendaylight_Tutorial/distribution/opendayli
al/distribution/packaging/1.0.0-SNAPSHOT/packaging-1.0.0-SNAPSHOT.tar.gz
[INFO] Installing /home/ubuntu/SDNHub_Opendaylight_Tutorial/distribution/opendayli
distribution/packaging/1.0.0-SNAPSHOT/packaging-1.0.0-SNAPSHOT.zip
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building main 0.6.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.6.1:clean (default-clean) @ main ---
[INFO]
[INFO] --- maven-enforcer-plugin:1.3.1:enforce (enforce-maven) @ main ---
[INFO]
[INFO] --- maven-checkstyle-plugin:2.14:check (default) @ main ---
[INFO]
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ main ---
[INFO] Installing /home/ubuntu/SDNHub_Opendaylight_Tutorial/pom.xml to /home/ubunt
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO]
[INFO] SDN Hub Tutorial project common properties ......... SUCCESS [  5.385 s]
[INFO] SDN Hub tutorial project common utils .............. SUCCESS [  7.397 s]
[INFO] learning-switch-parent ............................. SUCCESS [  0.057 s]
[INFO] SDN Hub tutorial project learning switch Impl ...... SUCCESS [ 14.106 s]
[INFO] SDN Hub tutorial project Learning Switch Config .... SUCCESS [  1.004 s]
[INFO] SDN Hub tutorial project Tap application Model ..... SUCCESS [  3.570 s]
[INFO] tapapp-parent ...................................... SUCCESS [  0.038 s]
[INFO] SDN Hub tutorial project Tap application Impl ...... SUCCESS [  8.533 s]
[INFO] SDN Hub tutorial Project Tap application Config .... SUCCESS [  0.167 s]
[INFO] SDN Hub tutorial project features .................. SUCCESS [  6.301 s]
[INFO] distribution-parent ................................ SUCCESS [  0.049 s]
[INFO] SDN Hub tutorial project Karaf branding ............ SUCCESS [  0.390 s]
[INFO] SDN Hub tutorial project distribution packaging .... SUCCESS [01:21 min]
[INFO] main ............................................... SUCCESS [  4.353 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 02:19 min
[INFO] Finished at: 2016-06-15T09:41:10-07:00
[INFO] Final Memory: 94M/349M
[INFO] ------------------------------------------------------------------------
ubuntu@sdnhubvm:~/SDNHub_Opendaylight_Tutorial[09:41] (master)$
```

Figura 49: Montaje del Proyecto Exitoso (Laboratorio 4)

(Fuente: Elaborada para este TFG)

Ahora que tenemos nuestro proyecto (en el cual se incluye el código java y los paquetes de la aplicación a instalar) montado, podemos arrancar el controlador. Para ello, lo hacemos de manera similar al de la versión de usuario, aunque el archivo para arrancarlo estará en una ruta:

```
cd distribution/opendaylight-karaf/target/assembly
```

```
./bin/karaf
```

Esperamos a que se carguen todos los módulos del controlador y aparecerá la interfaz de línea de comandos de Karaf con la tipografía de SdnHub.

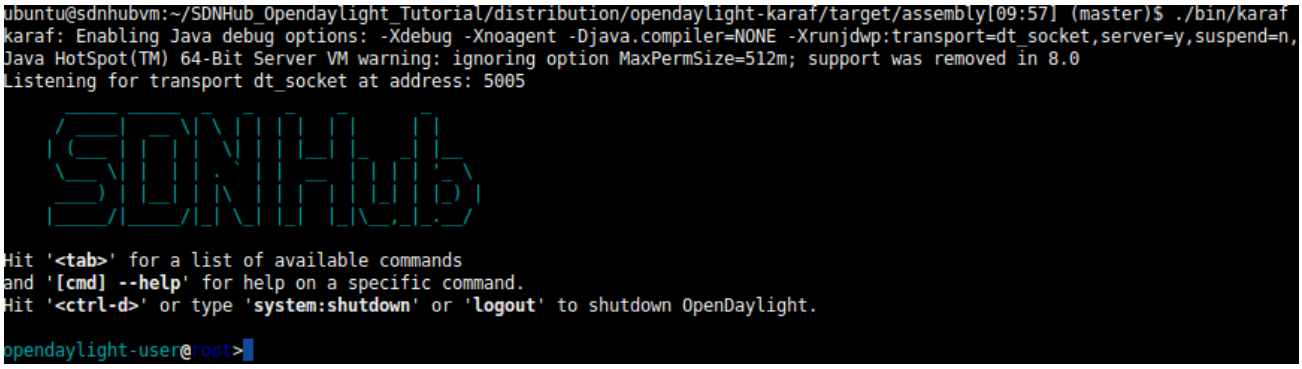


Figura 50: Interfaz de Línea de Comandos OpenDaylight (Laboratorio 4)

(Fuente: Elaborada para este TFG)

Si buscamos nuestra aplicación veremos que se encuentra disponible para ser instalada:

```
feature:list | grep sdnhub-tutorial-learning-switch
```

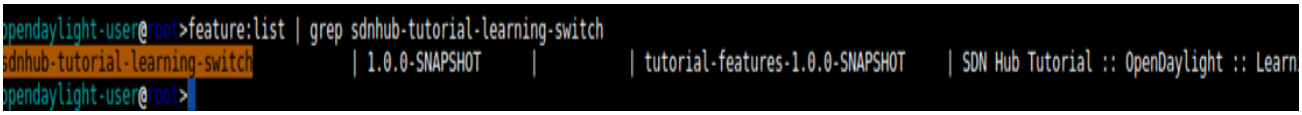


Figura 51: Búsqueda de la aplicación SDN a instalar (Laboratorio 4)

(Fuente: Elaborada para este TFG)

Ahora que ya tenemos nuestro controlador operativo, montamos la arquitectura.

Si intentamos hacer ping, observaremos que nos es imposible ya que aún no hemos instalado nuestra aplicación. Por lo tanto procedemos a ello:

```
feature:install sdnhub-tutorial-learning-switch
```

No obstante, con esto no es suficiente para tener ping. Debemos añadir un flujo manualmente para que nuestro switch reenvíe los paquetes al controlador:

```
sudo ovs-ofctl add-flow br0 priority=1,action=output:controller
```

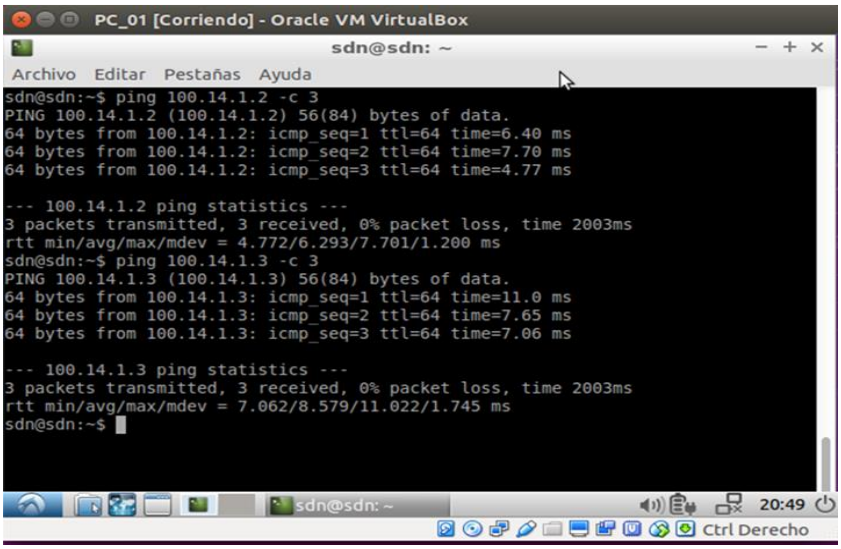


Figura 52: Ping con alto tiempo de respuesta (Laboratorio 4)

(Fuente: Elaborada para este TFG)

Podemos comprobar que ahora si tenemos conectividad entre máquinas. Sin embargo, este es más lento de lo habitual (ya que en un escenario tan sencillo debería estar en torno a 1 milisegundo). Este retardo se debe a que nuestro controlador únicamente actúa como un simple Hub y, por lo tanto, cada vez que un paquete llega al switch, este lo reenvía al controlador (mediante un mensaje OpenFlow Packet_IN), que le dice qué hacer con él (mediante un mensaje OpenFlow de tipo Packet_OUT). De hecho, si miramos la tabla de flujo del switch vemos como sólo dispone de la entrada que se añadió manualmente.

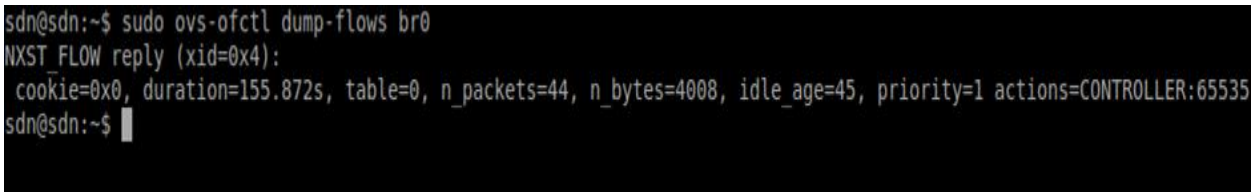


Figura 53 Tabla Flujo con Controlador funcionando en modo HUB (Laboratorio 4)

(Fuente: Elaborada para este TFG)

Nuestro objetivo por tanto, es programar con ayuda de los archivos facilitados por SdnHub un switch funcional. Para programar una aplicación en OpenDaylighy, se utiliza generalmente una vista de Modelo-Vista-Controlador donde se usa YANG como modelo de datos, la API de REST como vista (accesible a través de RESTconf) y una aplicación implementada generalmente en Java para manejar los eventos cuando se produce un cambio.

Vamos por tanto a navegar un poco por el código para procurar entenderlo y asimilarlo. Así pues abrimos el fichero TutorialL2Forwarding.java disponible en el paquete

"src/main/java" de nuestro proyecto. La lógica de nuestro switch viene marcada por el diagrama de flujo que podemos encontrar en la figura 49.

El siguiente código hace referencia a la lógica utilizada por el controlador y forma parte de un método que se activa cada vez que se recibe un paquete:

```
//Extraemos las cabeceras
byte[]etherTypeRaw =
PacketParsingUtils.extractEtherType(notification.getPayload());
int etherType = (0x0000ffff & ByteBuffer.wrap(etherTypeRaw).getShort());

//Si es un tipo de paquete LLDP (Link Layer Discover Protocol) lo ignoramos
if (etherType == 0x88cc) {
            return;
} else {
      //Almacenamos los datos del paquete y las direcciones MAC de //origen y
      destino
      byte[] payload = notification.getPayload();
      byte[] dstMacRaw = PacketParsingUtils.extractDstMac(payload);
      byte[] srcMacRaw = PacketParsingUtils.extractSrcMac(payload);
     //Pasamos las cadenas de bytes a Strings
      String srcMac = PacketParsingUtils.rawMacToString(srcMacRaw);
      String dstMac = PacketParsingUtils.rawMacToString(dstMacRaw);

      //Actualizamos la tabla, añadiendo MAC de origen e ID del puerto de
      //entrada
       this.macTable.put(srcMac, ingressNodeConnectorId);

        //Buscamos coincidencia en la tabla para determinar el puerto de
        //salida
       NodeConnectorId egressNodeConnectorId = this.macTable.get(dstMac) ;

      //Si la hubo
       if (egressNodeConnectorId != null) {

            //Añadimos un flujo al switch (o lo actualizamos en caso de
            //existir)
            programL2Flow(ingressNodeId, dstMac, ingressNodeConnectorId,
egressNodeConnectorId);
               NodeConnectorRef egressNodeConnectorRef =
InventoryUtils.getNodeConnectorRef(egressNodeConnectorId);
            //Enviamos el paquete por el puerto correspondiente
               packetOut(ingressNodeRef, egressNodeConnectorRef, payload);
      }else {
            //Reenviar el paquete por todos los puertos menos el de entrada
            packetOut(ingressNodeRef, floodNodeConnectorRef, payload);
           }
        }
    }
```

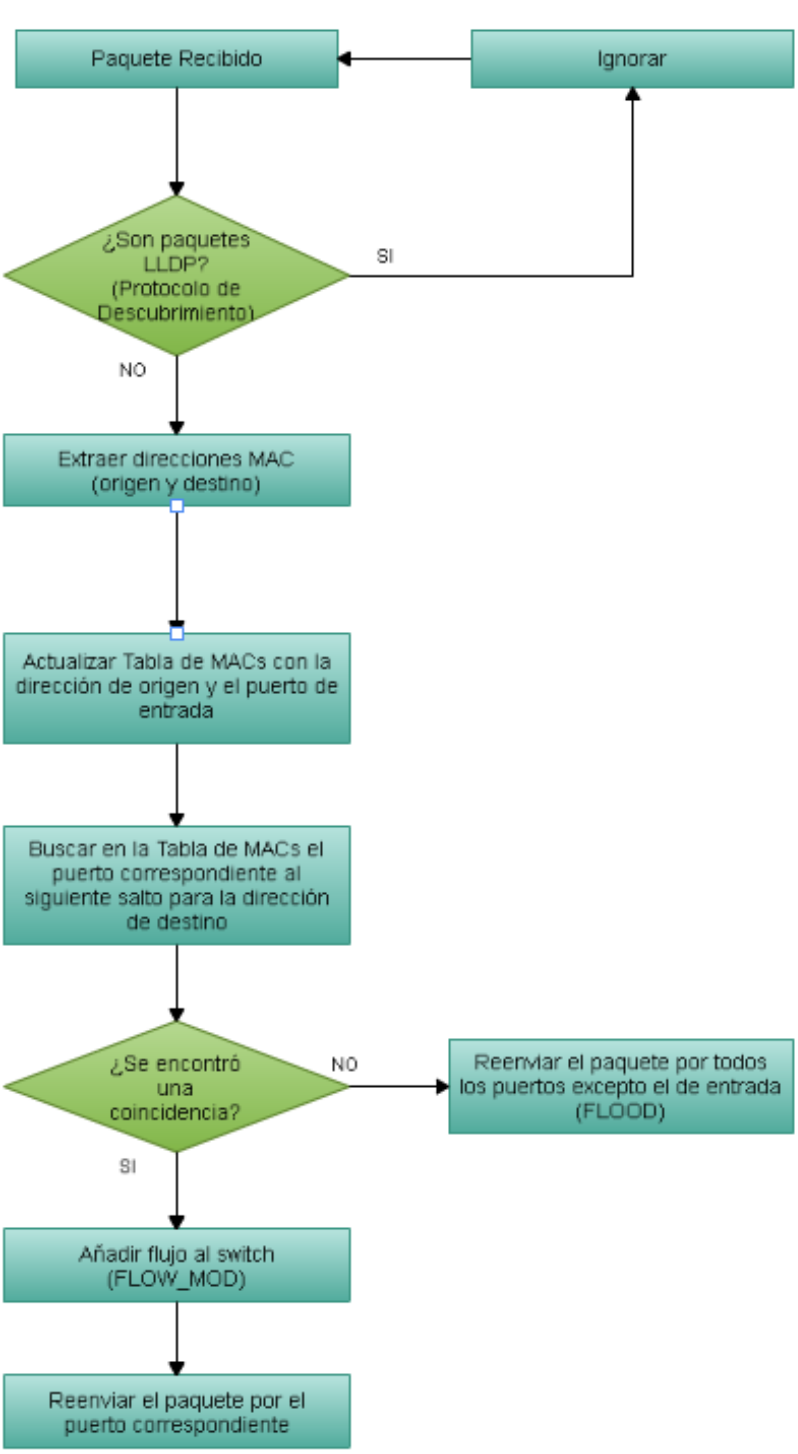


Figura 54: Diagrama de Flujo de la lógica de un Learning Switch (Laboratorio 4)

(Fuente: Elaborada para este TFG)

Por último sustituimos el antiguo fichero java, volvemos a compilar nuestro proyecto y arrancar el controlador. Si ejecutamos el ping ahora, comprobaremos como la primera vez sigue siendo más lento, pero luego aumenta la velocidad de respuesta. Esto es porque el controlador instala las entradas de flujo correspondientes en el switch y este ya no tiene que mandar un Packet_IN para cada flujo.

```
sdn@sdn:~$ sudo ovs-ofctl dump-flows br0
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=135.347s, table=0, n_packets=45, n_bytes=4334, idle_age=39, in_port=1,dl_dst=00:00:00:00:00:02 actions=output:2
 cookie=0x0, duration=35.284s, table=0, n_packets=10, n_bytes=942, idle_age=26, in_port=3,dl_dst=00:00:00:00:00:01 actions=output:1
 cookie=0x0, duration=34.314s, table=0, n_packets=10, n_bytes=942, idle_age=26, in_port=1,dl_dst=00:00:00:00:00:03 actions=output:3
 cookie=0x0, duration=152.258s, table=0, n_packets=4, n_bytes=392, idle_age=35, priority=1 actions=CONTROLLER:65535
```

Figura 55: Flujos añadidos por la Aplicación SDN (Laboratorio 4)

(Fuente: Elaborada para este TFG)

```
sdn@sdn:~$ ping 100.14.1.3 -c 10
PING 100.14.1.3 (100.14.1.3) 56(84) bytes of data.
64 bytes from 100.14.1.3: icmp_seq=1 ttl=64 time=21.4 ms
64 bytes from 100.14.1.3: icmp_seq=2 ttl=64 time=17.6 ms
64 bytes from 100.14.1.3: icmp_seq=3 ttl=64 time=0.865 ms
64 bytes from 100.14.1.3: icmp_seq=4 ttl=64 time=0.930 ms
64 bytes from 100.14.1.3: icmp_seq=5 ttl=64 time=0.596 ms
64 bytes from 100.14.1.3: icmp_seq=6 ttl=64 time=0.604 ms
64 bytes from 100.14.1.3: icmp_seq=7 ttl=64 time=0.671 ms
64 bytes from 100.14.1.3: icmp_seq=8 ttl=64 time=1.13 ms
64 bytes from 100.14.1.3: icmp_seq=9 ttl=64 time=0.978 ms
64 bytes from 100.14.1.3: icmp_seq=10 ttl=64 time=0.846 ms

--- 100.14.1.3 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9012ms
rtt min/avg/max/mdev = 0.596/4.574/21.441/7.542 ms
sdn@sdn:~$
```

Figura 56: Ping con Controlador funcionando como un Router de Capa 2 (Laboratorio 4)

(Fuente: Elaborada para este TFG)

5. GUION DE PRÁCTICAS

5.1 INTRODUCCIÓN

El desarrollo de este laboratorio tiene como objetivo principal que el alumno pueda asimilar los conceptos explicados en clase, aplicados a una arquitectura SDN real. Dado el alto volumen de temario abarcado, ni las arquitecturas ni los ejercicios de prueba propuestos requieren de configuraciones avanzadas.

Hemos dividido el guion en cuatro laboratorios:

1. **Laboratorio 1**: En este laboratorio se estudiarán diferentes tipos de entradas OpenFlow y su impacto en la red.

2. **Laboratorio 2**: Este escenario servirá para tener una visión periférica de las opciones ofrecidas por un controlador SDN, que en nuestro caso será OpenDaylight.

3. **Laboratorio 3**: En este apartado, configuraremos un balanceador de carga siguiendo un algoritmo de Round-Robin.

4. **Laboratorio 4**: Por último, veremos como instalar una aplicación java en nuestro controlador, y estudiaremos su funcionamiento.

Este laboratorio esta pensado para ser realizado en grupos de trabajo de 2 alumnos.

Para la corrección de la práctica siga el orden establecido y diríjase al profesor para su pertinente evaluación.

Duración total estimada: 4-6 horas

5.2 ENTORNO DE TRABAJO

Durante el desarrollo de esta práctica, se facilitará un equipo del laboratorio y 4 máquinas virtuales, que podrán ser clonadas tantas veces como sea necesario, para el despliegue de la arquitectura SDN requerida.

Los diferentes tipos de máquinas virtuales de los que se dispone son:

- **PC_XX:** Máquina virtual que simulará las funciones de un PC convencional y que utilizaremos como cliente o servidor.

- **OVS_XX:** Máquina virtual que ejercerá las labores de un switch OpenFlow.

- **ODL Basic Controller:** Máquina virtual cuya función será ejercer el papel de controlador SDN y que contará con una distribución de OpenDaylight (versión de usuario).

- **ODL Aplicación SDN:** Máquina virtual cuya función será ejercer el papel de controlador SDN y que contará con una distribución de OpenDaylight (versión de desarrollador).

Para ejecutar las máquinas virtuales utilizaremos Virtual Box.

5.3 INSTRUCCIONES DE CONFIGURACIÓN

En esta sección se aportan las características de las máquinas virtuales, y se dan una serie de directrices para poder configurarlas correctamente

5.3.1 PC_XX

DESCRIPCIÓN

- Esta VM tiene como S.O Linux. Está pensada para ser utilizada como Cliente/Servidor dentro de una topología de red sencilla.

- El usuario habilitado es `"sdn"` y la contraseña `"sdn1234"`.

- Además de las aplicaciones instaladas en la distribución base de Lubuntu, hemos instalado wireshark e iperf para la comodidad del usuario.

CONFIGURACIÓN Y USO

A continuación se detallarán los pasos que el alumno deberá seguir cada vez que necesite hacer uso de una nueva máquina virtual de tipo PC_XX:

1. **CLONACIÓN**: En primer lugar, y con la máquina apagada se procederá a la clonación de la VM PC_XX renombrándola con un nombre identificativo, y marcando la casilla de "Reinicializar la dirección MAC de todas las tarjetas de red".

2. **HABILITAR INTERFAZ DE RED:** Para poder habilitar la interfaz de red que se enlazará con la del switch OpenFlow , es indispensable disponer de un bridge en el host anfitrión. Este, servirá de enlace entre las máquinas virtuales haciendo las funciones de hub.

   En caso de no disponer de dicho bridge, lo podemos crear usando los siguientes comandos (los comandos indicados han sido probados bajo la versión de escritorio de Ubuntu 15.10) :

   ```
   sudo brctl addbr [nombre_bridge]
   sudo ip link set [nombre_bridge] up
   ```

   Una vez creados, configuramos nuestra interfaz en modo "Adaptador Puente" (bridge) y habilitamos el "Modo Promiscuo" seleccionando la opción "Permitir MVs". De lo contrario, los paquetes provenientes de otras máquinas serán descartados en el bridge.

3. **CONFIGURACIÓN IP:** Para añadir una dirección IP a la interfaz habilitada, podemos hacerlo de manera manual(`ifconfig`) o modificando el fichero de configuración de interfaces (ruta: `/etc/network/interfaces`). De este modo, el cambio será persistente aunque apaguemos/reiniciemos nuestra máquina virtual. Para ello debemos añadir los siguientes campos al fichero de configuración:

```
auto <nombre interfaz>
iface <nombre interfaz> inet static
address <dirección IP>
netmask <máscara de red>
broadcast <dirección broadcast>
```

5.3.2 OVS_XX

DESCRIPCIÓN

- Esta VM tiene como S.O Linux. Está pensada para ser utilizada como switch OpenFlow dentro de una topología de red sencilla, mediante el uso del software (ya instalado) de OpenVirtual Switch.

- El usuario habilitado es `"sdn"` y la contraseña `"sdn1234"`.

- Además de las aplicaciones instaladas en la distribución base de Lubuntu, hemos instalado wireshark e iperf para la comodidad del usuario.

CONFIGURACIÓN Y USO

A continuación se detallarán los pasos que el alumno deberá seguir cada vez que necesite hacer uso de una nueva máquina virtual de tipo OVS_XX:

1. **CLONACIÓN**: Proceso análogo al de la máquina PC_XX (consultar documentación).

2. **HABILITAR INTERFAZ DE RED**: Proceso análogo al de la máquina PC_XX (consultar documentación). Configurar tantas interfaces como sean necesarias.

   ****NOTA :*** Si necesita configurar más de 4 interfaces, puede hacerlo por línea de comandos. A continuación, se muestran los comandos utilizados para configurar una quinta interfaz en modo bridge:

```
VBoxManage modifyvm <nombreMV> --nic5 bridged
VBoxManage modifyvm <nombreMV> --bridgeadapter5 "<nombreBridge>"
```

3. **CONFIGURACIÓN IP:** Proceso análogo al de la máquina PC_XX (consultar documentación). Configurar tantas interfaces como sean necesarias.

   *****NOTA :*** Los enlaces que conectan a un switch con un PC no necesitan de dirección IP.

4. **CREACIÓN DEL SWITCH:** Para crear un switch y añadir interfaces al mismo deberemos utilizar los siguientes comandos:

   ```
   sudo ovs-vsctl add-br <nombre_bridge>
   sudo ovs-vsctl add-port <nombre_bridge> <nombre_interfaz>
   ```

   Y en caso de ser necesario:

   ```
   sudo ovs-vsctl set-controller <bridge> tcp:<ip_controlador>:6633
   ```

5.3.3 ODL BASIC CONTROLLER

DESCRIPCIÓN

- Esta VM tiene como S.O Linux. Está pensada para ser utilizada como controlador SDN.
- Dispone de una distribución del controlador de OpenDaylight.
- El usuario habilitado es `"sdn"` y la contraseña `"sdn1234"`.
- Además de las aplicaciones instaladas en la distribución base de Lubuntu, hemos instalado wireshark e iperf para la comodidad del usuario.

CONFIGURACIÓN Y USO

Para utilizar esta máquina debemos configurarla exactamente igual que las máquinas de tipo PC_XX.

Las especificaciones técnicas de la máquina virtual, podemos dejarlas como están o aumentarlas (siendo recomendables 4 núcleos y 4 GB de RAM). No obstante, para el desarrollo de este laboratorio se ha probado y ejecutado correctamente con las especificaciones establecidas. Si experimenta alguna ralentización, no dude en acomodar la máquina a sus necesidades.

Se recomienda el uso de una versión de Virtual Box igual o superior a la 5.0.0 para evitar problemas con la interfaz gráfica de usuario de ODL.

5.3.4 ODL APLICACIÓN SDN

DESCRIPCIÓN

- Máquina virtual desarrollada por el grupo SdnHub. Tiene como S.O Linux. Está pensada para ser utilizada como controlador SDN.
- Dispone de una distribución del controlador de OpenDaylight para desarrolladores además de otras funcionalidades que no utilizaremos.
- El usuario habilitado es `"ubuntu"` y la contraseña `"ubuntu"`.
- Además de las aplicaciones instaladas en la distribución base de Lubuntu, hemos instalado wireshark e iperf para la comodidad del usuario.

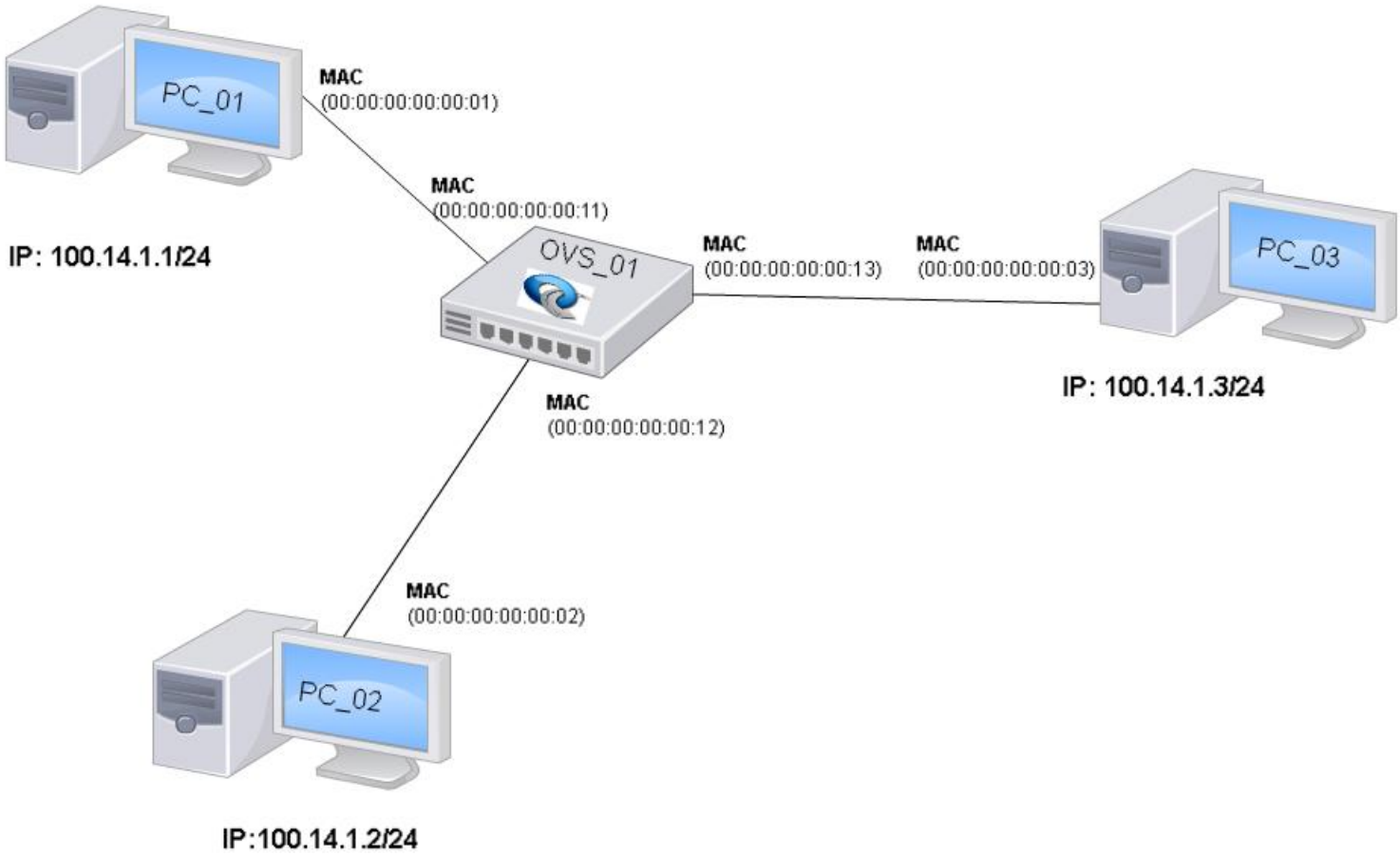
CONFIGURACIÓN Y USO

Para utilizar esta máquina debemos configurarla exactamente igual que las máquinas de tipo PC_XX.

5.4 LABORATORIOS

5.4.1 LABORATORIO 1 : ENTRADAS DE FLUJO OPENFLOW EN OPEN VIRTUAL SWITCH (2 horas)

Para la realización de este laboratorio, será necesario montar la siguiente arquitectura siguiendo las directrices de configuración marcadas en el apartado 5.3.



Para comprobar el estado del bridge, podemos usar los siguientes comandos. El primero, mostrará una breve estructura de todos los switches creados usando OVS y el segundo, aportará además información específica de las interfaces añadidas:

```
sudo ovs-vsctl show
sudo ovs-ofctl show <nombre-bridge>
```

PARTE I: ROUTING BÁSICO

Antes de comenzar es recomendable ejecutar el siguiente comando en una terminal de la máquina virtual de nuestro switch:

```
sudo ovs-ofctl del-flows <nombre-bridge>
```

De este modo, eliminaremos cualquier flujo que pudiese haber en nuestro switch. Se recomienda ejecutar este comando cada vez que pasemos a una nueva parte, salvo que se indique lo contrario.

Hito 1. Ejecutar un PING desde cualquiera de las máquinas y analizar el resultado.

Añadimos ahora manualmente nuestro primer flujo al switch mediante el siguiente comando:

```
sudo ovs-ofctl add-flow <nombre-bridge> action=normal
```

Para comprobar si se ha añadido correctamente, usamos el siguiente comando, el cual nos muestro los flujos instalados en el switch:

```
sudo ovs-ofctl dump-flows <nombre-bridge>
```

Hito 2.1 Comprobar conectividad entre máquinas.

Hito 2.2 Si vuelves a mostrar ahora los flujos del switch, ¿qué diferencias observas? Analiza brevemente los campos de la entrada de flujo mostrada.

PARTE II: CONCORDANCIA DE CAPA 2

En este introduciremos ya flujos algo más específicos. Por ello partiendo de la arquitectura anterior borramos los flujos del switch y añadimos los nuevos:

```
sudo ovs-ofctl add-flow br0 priority=500,in_port=1,actions=output:2
sudo ovs-ofctl add-flow br0 priority=500,in_port=2,actions=output:1
```

Hito 3. Comprobar conectividad entre máquinas y justificar el resultado.

A continuación, y manteniendo los flujos añadidos, instalamos una tercera entrada en nuestro switch:

```
sudo ovs-ofctl add-flow br0 priority=32768,actions=drop
```

Hito 4. Comprobar conectividad entre máquinas y justificar el resultado.

Para el siguiente hito, vamos a proceder a indicar a nuestro switch un comportamiento similar, de manera que pueda establecerse una comunicación entre los equipos 1 y 2, pero esta vez tomando como condición de coincidencia las direcciones MAC de origen y destino.

Elimina todos los flujos anteriores y añade las siguientes entradas:

```
sudo ovs-ofctl add-flow br0
dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,actions=output:1

sudo ovs-ofctl add-flow br0
dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01,actions=output:2
```

Hito 5.1 Comprobar conectividad entre máquinas y justificar el resultado.

Recomendación: Capturar los paquetes generados en wireshark puede ayudarte a comprender la situación.

Añade el siguiente flujo al switch y comprueba su efecto:

```
sudo ovs-ofctl add-flow br0 dl_type=0x806,nw_proto=1,actions=flood
```

Hito 5.2 Analiza la entrada añadida y su funcionalidad.

Hito 5.3 Añadir las entradas que estime necesario para tener conectividad entre todas las máquinas virtuales.

PARTE III: CONCORDANCIA A NIVEL IP

En el apartado anterior nuestro switch OpenFlow utilizaba puertos y direcciones MAC para tomar las decisiones de reenvío. A continuación, añadiremos una serie de entradas para que la concordancia se realice a nivel IP.

Eliminamos todos los flujos y añadimos los siguientes:

```
sudo ovs-ofctl add-flow br0
priority=500,ip,nw_src=100.14.1.0/24,nw_dst=100.14.1.0/24,actions=norm
al
sudo ovs-ofctl add-flow br0 arp,nw_dst=100.14.1.1,actions=output:1
sudo ovs-ofctl add-flow br0 arp,nw_dst=100.14.1.2,actions=output:2
sudo ovs-ofctl add-flow br0 arp,nw_dst=100.14.1.3,actions=output:3
```

Hito 6.1 Comprobar conectividad entre los equipos

Hito 6.2 Explicar las diferencias de funcionamiento con la entrada de flujo añadida en el hito 5.2. ¿Cuál es más ventajoso? Razona tu respuesta.

Otra de las ventajas de poder utilizar entradas OpenFlow es que nos permite diferenciar tráficos de forma fácil y sencilla. Para nuestro escenario, añadiremos una entrada que diferencie el tipo de servicio modificando el valor del campo DSCP en los paquetes con origen PC_03. Le daremos un valor (elegido de manera arbitraria) de 46.

De modo que, si:

$$ToS = DSCP_{(6\ bits)} + ENC_{(2\ bits)}$$

Y queremos un valor de DSCP = 46 (101110 en binario):

$$ToS = 10111000_{bin} = 184_{dec}$$

```
sudo ovs-ofctl add-flow br0
priority=X,ip,nw_src=100.14.1.3,actions=mod_nw_tos:184,normal
```

Hito 7.1 Dar un valor adecuado a la prioridad de la entrada de flujo anterior para obtener el funcionamiento esperado.

Hito 7.2 Comprobar que se modifica el campo DSCP en los paquetes provenientes del PC_03.

PARTE IV: FIREWALL TCP

Para esta parte, vamos a programar nuestro switch para actuar como un firewall, de manera que filtrará los paquetes dirigidos al PC_03.

Partiendo de una tabla de flujo vacía, añadimos las entradas que aseguraran el correcto funcionamiento de la red entre los clientes y el servidor:

```
sudo ovs-ofctl add-flow br0 arp,actions=normal

sudo ovs-ofctl add-flow br0
priority=800,nw_src=100.14.1.3,actions=normal

sudo ovs-ofctl add-flow br0
priority=500,dl_type=0x800,nw_proto=6,tp_dst=80,actions=output:3
```

Hito 8.1 Analizar las entradas introducidas y su función en nuestro escenario.

Hito 8.2 Explicar el significado de los valores de los campos dl_type, nw_proto y tp_dst de la última entrada.
Hito 8.3 Comprobar conectividad y justificar el resultado.

Lanzamos el servidor en el PC_03 haciendo uso del siguiente comando:

```
iperf -s -p 80
```

Desde cualquiera de los otros equipos ejecutamos un cliente :

```
iperf -c -p 80
iperf -c -p 90
```

Hito 8.4 Analizar el comportamiento observado y justificarlo.

5.4.2 LABORATORIO 2 : OPERACIONES BÁSICAS CON OPENDAYLIGHT (1 hora)

Para esta parte del laboratorio puedes utilizar las máquinas que ya configuraste para el Laboratorio 1, acordándote de borrar las entradas restantes en el switch y añadiendo una interfaz más de red que conectará el switch con nuestro controlador (utilizaremos como controlador la máquina "ODL Basic Controller" y la distribución habilitada para dicho laboratorio).

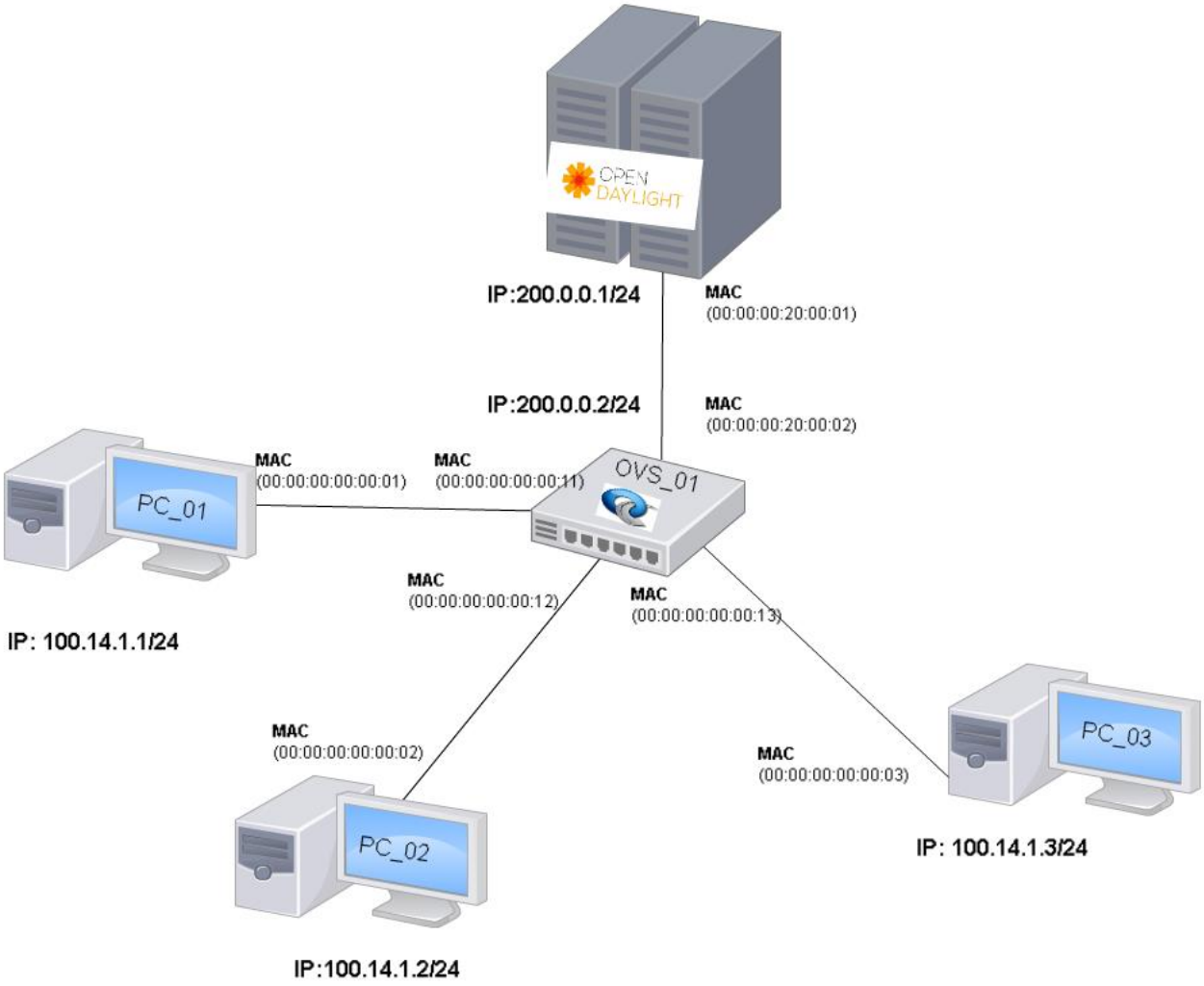


Figura 2: Topología Laboratorio 2

El objetivo de esta práctica es familiarizarse con el entorno bajo el que trabaja OpenDaylight, así como asimilar la función de un controlador dentro de una topología de red SDN.

El objetivo planteado es familiarizarnos con el entorno de OpenDaylight (tanto en su interfaz de línea de comandos como en su interfaz gráfica web) y entender los métodos que podemos utilizar para sacarle provecho.

Para arrancar el controlador tenemos diferentes formas de hacerlo (estando en la carpeta donde tenemos la distribución):

- **bin/karaf** → arranca el controlador y la consola de karaf.
- **bin/start** → arranca el controlador en segundo plano.
- **bin/stop** → para el controlador.
- **bin/client -u karaf** → arranca consola karaf cuando el controlador está en ejecución.

Así pues, arrancamos Karaf (interfaz de línea de comandos de OpenDaylight) y nos aparecerá una consola con el grafo de ODL:

Figura 3: Consola Karaf

Las distribuciones de ODL vienen por defecto sin ninguna característica instalada. Pero podemos fácilmente añadirlas sin necesidad de una conexión a internet.

A continuación puede ver una serie de comandos que serán útiles a la hora de trabajar con Karaf:

- **feature:list (-i)** → Listado de (o solo las instaladas si añadimos la opción '-i') las features disponibles en el controlador.
- **feature:info <feature>** → Muestra información relevante de la feature requerida.
- **feature:install <feature>** → Instala la feature indicada.
- **bundle:list -s** → Listado de los bundles cargados.

Este es el método para la instalación mediante Karaf. También podemos añadir features a nuestro controlador especificandolas en el archivo de configuración (`<carpeta-distribucion>/etc/org.apache.karaf.features.cfg`) antes de arrancarlo. Debes añadir las features que quieras a continuación de las especificadas en la línea:

```
featuresBoot = config,standard,region,package,kar,ssh,management
```

Hito.9 Modifica el archivo de configuración para instalar la feature "l2switch" (utiliza los comandos facilitados para averiguar su nombre exacto y comprobar que se ha instalado correctamente).

Con el controlador arrancado podemos acceder a la interfaz gráfica (conocida como dlux) en http://<IP_controlador>:8181/index.html o si accedemos desde otra máquina virtual conectada al controlador, sustituyendo localhost por la IP del mismo).

Pero antes de intentar acceder, debemos cargar todas las features referentes a Dlux con el comando `"feature:install odl-dlux-all"`

Abrimos un navegador y accedemos a la URL indicada anteriormente. Nos aparecerá una pantalla para iniciar sesión donde los credenciales son "admin/admin".

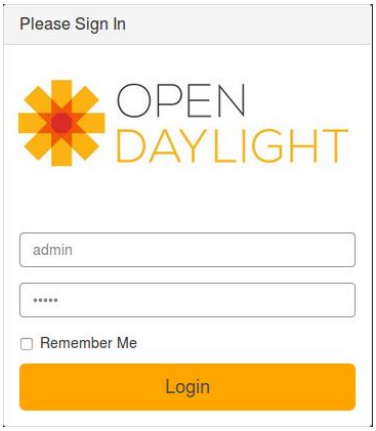


Figura 4: Pantalla de Inicio de Sesión de DLUX

Antes de continuar nos aseguramos de que el controlador ha cargado todos sus módulos correctamente con el comando: `"log:display"`, donde se indica cuando está 100% operativo .

Hito 9.1 Verifica que el controlador establece la conexión con el switch OpenFlow y analiza la topología.

Hito 9.2 Analiza la conectividad entre máquinas y explica el resultado. ¿Ha cambiado la topología? ¿Por qué?

Hito 9.3 Analiza y explica los flujos añadidos al switch.

A continuación, vamos a utilizar una interfaz REST para obtener la topología desde el punto de vista del controlador. Esta, es una interfaz con acceso directo a las configuraciones dentro de nuestro controlador desde el directorio la URL http://localhost:8181/restconf/* . Dentro de este, se diferencian dos sub-rutas:

- **restconf/config/*:** Aquí los usuarios (generalmente administradores de red) pueden crear, actualizar, consultar o eliminar (POST, PUT, GET, DELETE) la configuración de las aplicaciones.
- **restconf/operational/*:** Aquí es dónde las aplicaciones escriben los estados y donde los usuarios pueden consultarlos (GET). Las bases de datos más consultadas dentro de este directorio, son la topología y el inventario de nodos.

Podemos utilizar diferentes herramientas para acceder a estos directorios. Para este laboratorio vamos a utilizar en un primer lugar la herramienta disponible en Dlux YangUI (herramienta que permite buscar URLs y métodos del controlador).

Para ello, pinchamos en YangUI → opendaylight-inventory →operational→nodes. En el desplegable de abajo seleccionamos GET como método y presionamos en enviar

Hito 10. Analizar la respuesta obtenida.

De manera equivalente podemos consultar la topología usando el comando `"curl"` de Linux con el siguiente formato:

```
curl --user <user>:<password> -H <cabecera1> … -H <cabeceraN> -X<tipo de petición http> <url> -d '<cuerpo de la petición>'
```

```
Nota: Es indispensable añadir como mínimo las cabeceras "Accept" y "Content-type" las cuales aceptan tanto "json" como "xml" como formatos (Ejemplo de cabecera: "Accept:application/xml")
```

Hito 11. Formula la petición y analiza la respuesta.

5.4.3 LABORATORIO 3 : BALANCEADOR DE CARGA (1 hora)

Para este laboratorio podemos utilizar las mismas máquinas virtuales del apartado anterior añadiendo un equipo más. Como controlador utilizaremos la distribución disponible en la carpeta Laboratorio 3 de "ODL Basic Controller".

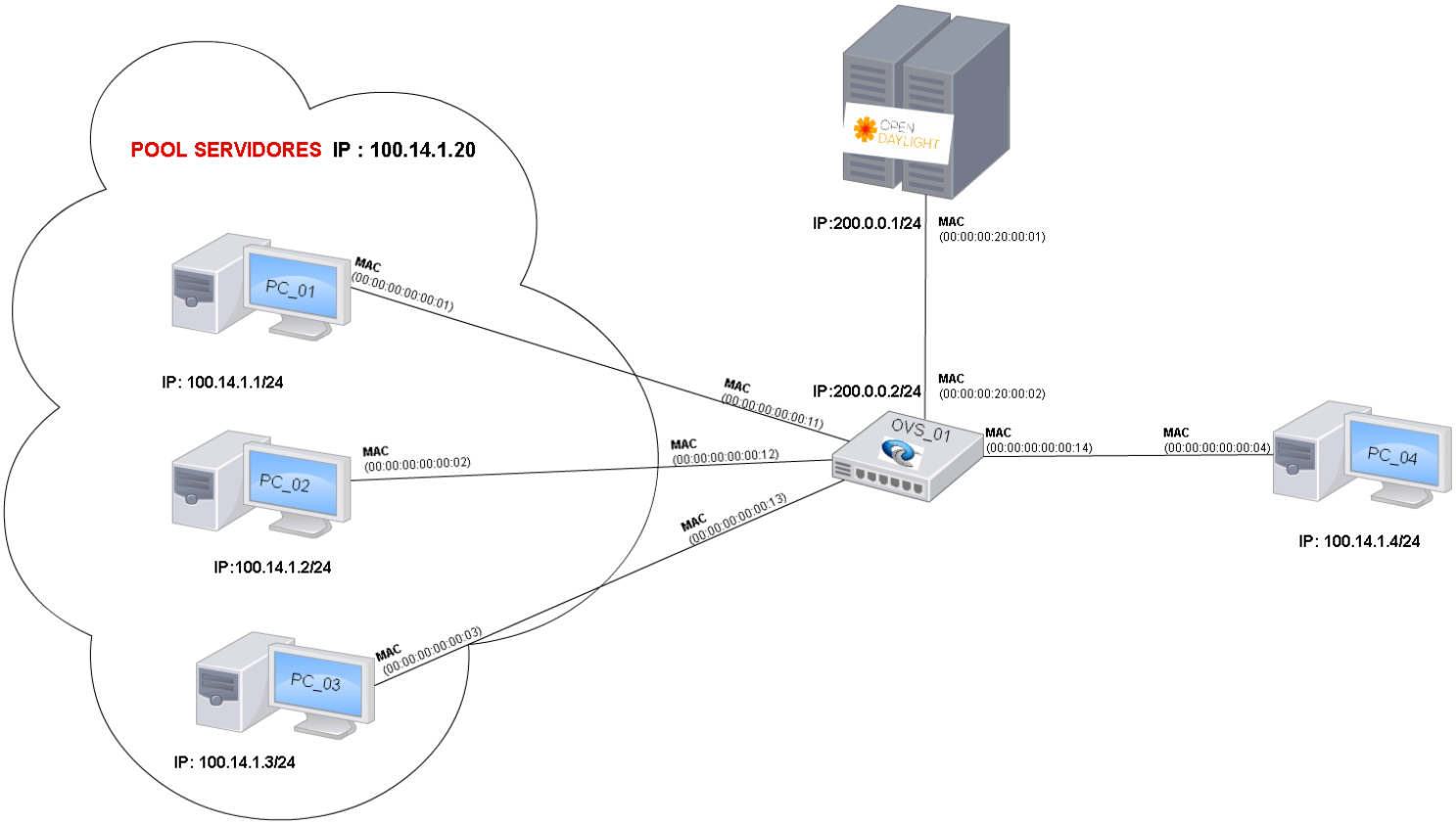


Figura 5: Topología Laboratorio 3

Una vez montada la arquitectura y con el controlador arrancado comprobamos la conectividad entre todos los equipos antes de continuar.

Configuraremos un balanceador de carga sencillo mediante una interfaz REST que utilizará un algoritmo de Round-Robin para enrutar las peticiones a los PCs 1, 2 y 3.

Para dicha implementación crearemos una cola de servidores que atenderán las peticiones dirigidas a una dirección IP virtual que crearemos. Para ello se disponen de los siguientes comandos:

#CREACIÓN DE UN NUEVO POOL

```
curl -- user "admin":"admin" -H "Accept:application/json" -H "Content-type:application/json" -X POST http://localhost:8080/one/nb/v2/lb/default/create/pool -d '{"name":"<nombre_pool>","lbmethod":"roundrobin"}'
```

#CREACIÓN DE UNA VIP

```
curl -- user "admin":"admin" -H "Accept:application/json" -H "Content-type:application/json" -X POST http://localhost:8080/one/nb/v2/lb/default/create/vip -d '{"name":"<nombre_VIP>","ip":"<IP>","protocol":"<protocolo>","port":"<puerto>","poolname":"<nombre_pool>"}'
```

#AÑADIR MIEMBRO AL POOL

```
curl -- user "admin":"admin" -H "Accept:application/json" -H "Content-type:application/json" -X PUT http://localhost:8080/one/nb/v2/lb/default/create/poolmember -d '{"name":"<nombre_Servidor>","ip":"<IP_Servidor>",poolname":"<nombre_pool>"}'
```

Hito 12. Utiliza los comandos aportados para configurar el pool y arranca los servidores en los PCs 1, 2 y 3.

Hito 13. Arranca wireshark y analiza qué ocurre si intentamos establecer una conexión desde el PC_04 a la IP virtual. ¿Por qué sucede esto? Modifica la configuración del PC_04 para solucionarlo.

Hito 14. Comprueba que las peticiones se enrutan bajo el algoritmo especificado y que se establece la conexión.

5.4.4 LABORATORIO 4 : INSTALACIÓN DE UNA APLICACIÓN JAVA EN NUESTRO CONTROLADOR (2 horas)

En este laboratorio se tiene como objetivo aprender a instalar una aplicación propia.

Dado lo laborioso del proceso y de que no se dispone de tiempo suficiente utilizaremos una sencilla aplicación desarrollada por el grupo SdnHub y que al igual que hacía la feature instalada en el Laboratorio 2 añadirá la lógica de un switch de capa 2 a nuestro controlador.

La arquitectura utilizada es la misma que en el Laboratorio 2 cambiando la máquina virtual del controlador por la de "ODL Aplicación SDN"

Para montar nuestra aplicación utilizaremos Maven (herramienta de software para la gestión y construcción de proyectos).

Iniciamos la máquina virtual del controlador y en la ruta del proyecto ejecutamos el siguiente comando:

```
mvn clean install -DskipTest -nsu
```

Si el proceso va bien, veremos en la pantalla un mensaje de éxito ("BUILD SUCCESS") que nos indicará que se ha realizado el montaje con éxito.

Con el proyecto correctamente montado arrancamos el controlador:

```
cd distribution/opendaylight-karaf/target/assembly
```

```
./bin/karaf
```

Esperamos a que se carguen todos los módulos del controlador y aparecerá la interfaz de línea de comandos de Karaf con la tipografía de SdnHub:

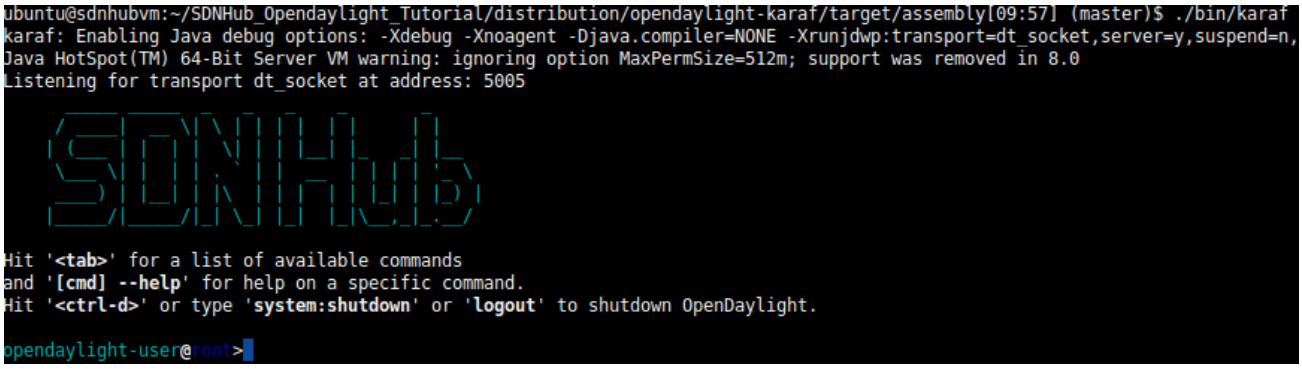


Figura 6: Interfaz de Línea de Comandos

Hito 15. Buscar e instalar nuestra aplicación utilizando los comandos aprendidos en el Laboratorio 2 de este guion de prácticas.

Hito 16. Añadir una entrada de flujo en nuestro switch para que reenvíe los paquetes al controlador.

Hito 17. Comprobar conectividad entre los equipos.

Si observamos los tiempos de respuesta del ping y lo comparamos con ejercicios anteriores podemos observar que es sustancialmente mayor.

Hito 18. Captura los paquetes con wireshark, analiza su recorrido y razona por qué tardan más en llegar.

Hito 19 (opcional). Modificar el código del archivo TutorialL2Forwarding.java (disponible en src/main/java) para que nuestro switch funcione como debería. Únicamente añadir el código necesario al método "onPacketReceived" de dicho archivo.

Utilice el siguiente diagrama de flujo como apoyo:

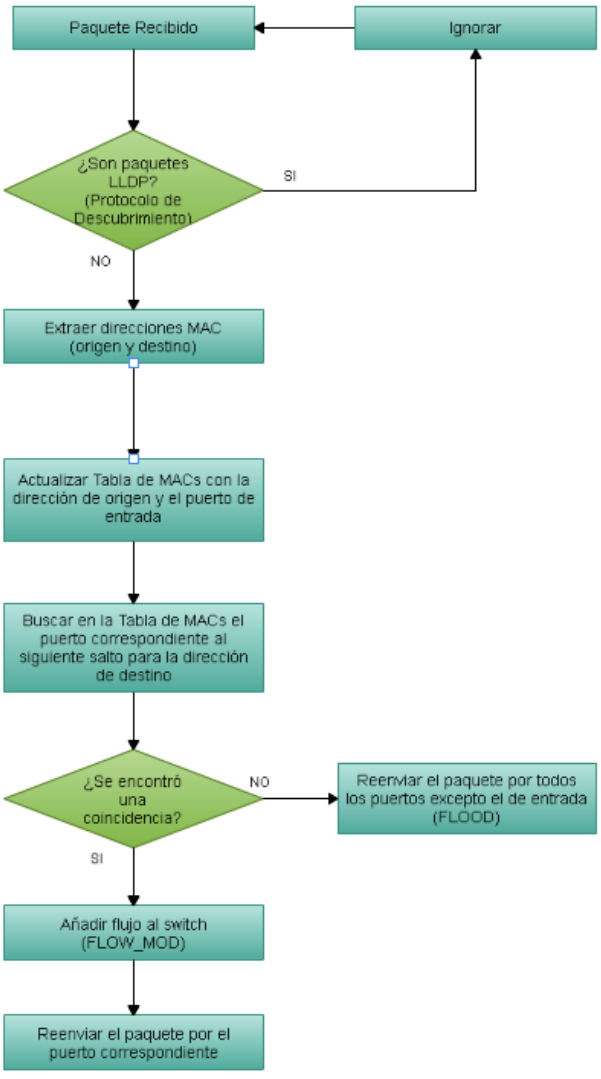


Figura 7: Diagrama Flujo Learning Switch

Si no ha realizado el hito 19 contacte al profesor para que le facilite el código de respuesta. Cualquier tipo de divulgación de dicho código, supondrá el suspenso automático del alumno de las prácticas.

Sustituya el código y vuelva a compilar el proyecto.

Hito 20. Arranque el controlador y vuelva a probar la conectividad entre los equipos. Justifique el comportamiento observado y compruebe la tabla de flujo del switch.

6. CONCLUSIONES Y LÍNEAS FUTURAS

En este punto, se analizará el trabajo realizado en cuanto a los objetivos marcados antes de iniciar el proyecto, así como posibles líneas de evolución para continuar con el mismo o mejorar algunos aspectos.

6.1 Conclusiones

Este Trabajo de Fin de Grado, tenía como objetivo primordial disponer de una arquitectura de red definida por software sobre la que poder trabajar y realizar diferentes pruebas.

Cuando se inició el proyecto, se determinó que la meta final sería obtener un guion de prácticas a poder ser utilizado en los laboratorios de la Universidad Carlos III de Madrid por alumnos de máster. Para su cumplimiento, se facilitaría al alumno (además del propio manual) las máquinas virtuales necesarias para poder desplegar las topologías de red.

Otro de los aspectos sobre los que se hizo especial hincapié fue la separación, es decir, cada máquina virtual (simulando un escenario real) no debía implementar más funciones que las cuales para las que había sido diseñado. Así pues, las máquinas se dividieron en tres tipos: PC, switches y controladores.

Teniendo estos objetivos en cuenta, se diseñaron, configuraron y probaron las máquinas virtuales adecuadamente, además de las diferentes topologías. Se concluyó, que sería más fácil para el alumno asimilar los conceptos más importantes de SDN, si se establecía una progresión clara empezando por casos sencillos e incrementando la dificultad a la vez que se añadían más elementos a la red.

Tras realizar diferentes pruebas, se ha llegado a la conclusión de que se han cumplido los objetivos marcados en un principio, pues se ha puesto a disposición de los alumnos, las herramientas necesarias para desplegar desde una arquitectura de red SDN sencilla, a una complicada, simplemente clonando las máquinas virtuales facilitadas. Así mismo, podemos añadir que para simular entornos de trabajo más extensos a los expuestos en este Trabajo de Fin de Grado sería necesario de un equipo anfitrión de altas prestaciones para garantizar una cierta fluidez.

Por último, y dado el gran abanico de opciones de configuración y programabilidad que ofrecen las redes definidas por software, podemos concluir que es una tecnología que pronto tendrá una gran repercusión en el mercado, siendo ampliamente importante que exista personal cualificado y con experiencia en estos entornos de trabajo. Mediante el

uso de los escenarios planteados en este proyecto, se podrá ayudar a formar a dichos especialistas.

6.2 Líneas Futuras

Como previamente comentábamos, las opciones para la administración de la red que nos otorga una arquitectura de red definida por software son difícilmente imaginables. Es por ello que se presentan muchas líneas sobre las que seguir trabajando:

- **Uso de otros Sistemas Operativos:** Cuando se despliega una arquitectura de red definida por software, es común recurrir a una distribución Linux por las facilidades de configuración que estas aportan. No obstante, y dado que se trata de una infraestructura de red se podrían plantear escenarios con clientes/servidores con diferentes sistemas operativos (por ejemplo: un cliente Linux, un servidor Windows y otro servidor Mac OS). Sin embargo, si quisiéramos utilizar otro sistema operativo para las máquinas referentes al switch y controlador, tendríamos que verificar la disponibilidad del software utilizado (OpenVirtualSwitch y OpenDaylight).

- **Portabilidad de las MVs:** Las máquinas virtuales han sido desarrolladas y configuradas bajo el uso de Virtual Box. Podríamos fácilmente portarlas para ser utilizadas con otros hipervisores (por ejemplo KVM es uno de los más utilizados en el despliegue de arquitecturas de red definidas por software).

- **Reajuste y mejora de las especificaciones técnicas:** Se podría hacer un estudio exhaustivo de rendimiento, para ajustar los requisitos técnicos de las máquinas virtuales dando la máxima productividad.

- **Despliegue de una arquitectura de red más amplia:** Se podría evaluar el comportamiento de la red empleando escenarios con más hosts, switches, varios controladores,… A modo de ejemplo, sería interesante observar cómo en un escenario mayor el controlador redirige el tráfico entre dos equipos al caerse un enlace.

- **Despliegue de una arquitectura mixta:** Sería interesante evaluar el funcionamiento de la red en situaciones de comunicación con otras arquitecturas convencionales.

- **Despliegue de un balanceador de carga con un algoritmo más complejo:** En este TFG se ha utilizado por simplicidad un mecanismo de Round-Robin, pero se podría desarrollar una aplicación SDN en java que automáticamente balanceara las peticiones a los servidores cuando (por ejemplo) se llegara a un 70% de congestión en el enlace.

- **Uso de la herramienta POSTMAN:** En este proyecto se ha utilizado brevemente las herramientas YangUI y el comando de Linux "curl" para realizar configuraciones a través de una interfaz REST. Podría utilizarse también la extensión de Chrome Postman para realizar dichas configuraciones.

- **Instalación de Flujos vía Rest API:** Utilización de la interfaz REST para programar flujos de manera proactiva.

- **Analizar el rendimiento con otros controladores SDN:** Utilizando herramientas como "cbench" o "Hcprobe" se podría evaluar el rendimiento de diferentes controladores en un mismo escenario.

6. CONCLUSIONS AND FUTURE WORKS (English Version)

In this chapter the achievement of the goals marked in the beginning of this project are discussed, as well as a range of possible future works and improvements.

6.1 Conclusions

In this Bachelor Thesis the main spot was focused into deploying a Software-Defined Networking architecture.

To start with, it was determined the final objective would be to use the architecture deployed to draw up guide notes for a laboratory. For this purpose, Carlos III University students would be provided not only with proper guidelines but also with virtual machines needed.

In the second place, separation of functions and virtual machines was emphasized. It was essential that every VM performed exclusively the tasks concerning their responsibilities. Therefore, they were categorized in three types: PCs, switches and controllers. Having this into account, virtual machines were properly designed, configured and tested. Besides, we came to the conclusion that establishing an ongoing scenario it would be easier for the students to understand the concepts of SDN.

After the study, we can conclude that the objectives marked at the starting point have been successfully completed. Students can deploy every topology they demand simply cloning VMs. Nevertheless, it is important to highlight the fact that a high performance guest host is desired if we want to deploy a large scenario.

Finally, due to the huge amount of possibilities in programmability and configuration allowed by Software-Defined Networking, we are able to conclude that they have a promising future and consequently, well-trained students will be demanded by the companies. Through the use of the guidelines exposed in this project, we are providing the university with tools to train these specialists.

6.2 Future Works

As mentioned before, lots of future works are possible. Here some further improvements of the system implemented or proposed changes are listed:

- **Using other Operative Systems:** When deploying a SDN architecture is usual the use of a Linux distribution. However, it would be interesting to observe the behavior in a network composed by several clients or servers running different OS –note that tools used to simulate OpenFlow switch and OpenDaylight controller are not available for all OS- .

- **Portability of VMs:** Virtual machines developed in this Bachelor Thesis have been intended to run under Virtual Box hypervisor. With some easy steps we could expand them in order to be able to use them in other hypervisors as KVM or VMware.

- **Readjustment and improvement of technical specifications:** Carrying out some performance tests we could determine the best technical specifications for high productivity.

- **Deploying a larger architecture:** Since we have used a simple topology, it would be interesting to observe the adaptation of the controller to a larger scenario and how it reacts to a broken connection.

- **Mixing topologies:** As we mentioned earlier in this project when describing types of OpenFlow switches, some hybrid switches are available. Therefore, it would be interesting to evaluate the functioning of a mixed topology.

- **Deploying a more complex load balancer:** In this project a load balancer running Round-Robin algorithm was implemented. An option of improvement would be to deploy a more complex method to forward TCP requests (e.g. routing requests when a link presents a 70% congestion grade).

- **Using POSTMAN tool:** Using Chrome Postman extension instead of YangUI or "curl" Linux command in order to carry out configuration tasks through a REST interface.

- **Installing flows via REST APIs:** Install flow entries in a proactive way using a REST interface.

- **Analyze performance of different SDN Controllers:** Use of tools –as "cbench" or "Hcprobe"- in order to analyze the performance of different controllers in the same scenario.

7. ANEXOS

En este capítulo se detalla la planificación del proyecto, los recursos utilizados y el presupuesto junto a información adicional que consideramos relevante.

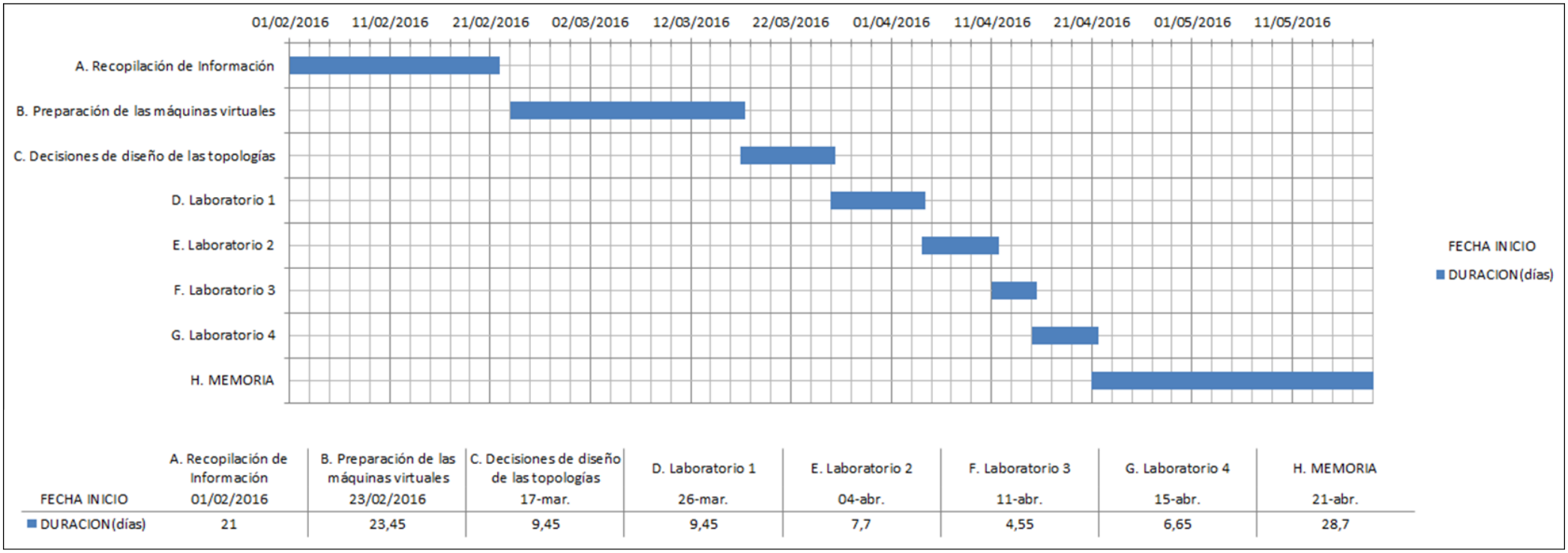
A. PLANIFICACIÓN DE TAREAS, RECURSOS Y PRESUPUESTO

A.1 PLANIFICACIÓN DEL TRABAJO DE FIN DE GRADO

TAREA	HORAS
A. Recopilación de Información	60
a.1 Estudio de conceptos de funcionamiento de SDN	25
a.2 Estudio de conceptos de OpenFlow	12
a.3 Estudio de conceptos sobre controladores SDN	11
a.4 Análisis y elección de controlador	12
B. Preparación de las máquinas virtuales	67
b.1 Configuración de la máquina PC_XX	15
b.2 Configuración de la máquina OVS_XX e instalación del software OVS	20
b.3 Configuración de la máquina ODL Basic Controller e instalación de OpenDaylight	20
b.4 Configuración de la máquina ODL Application SDN	12
C. Decisiones de diseño de las topologías	30
c.1 Estudio e implementación de los métodos de interconexión	20
c.2 Diseño de las topologías a utilizar	10
D. Laboratorio1	30
d.1 Despliegue de la arquitectura	13
d.2 Pruebas y redacción del guion	17
E. Laboratorio 2	25
e.1 Despliegue de la arquitectura	10
e.2 Pruebas y redacción del guion	15
F. Laboratorio3	15
f.1 Despliegue de la arquitectura	5
f.2 Pruebas y redacción del guion	10
G. Laboratorio4	20
g.1 Despliegue de la arquitectura	8
g.1 Pruebas y redacción del guion	12
H. Memoria	85
h.1 Redacción de la memoria	70
h.2 Preparación de la presentación	15
TOTAL	**332**

Tabla 1: Desglose de Tareas

A.2 DIAGRAMA DE GANTT



	A. Recopilación de Información	B. Preparación de las máquinas virtuales	C. Decisiones de diseño de las topologías	D. Laboratorio 1	E. Laboratorio 2	F. Laboratorio 3	G. Laboratorio 4	H. MEMORIA
FECHA INICIO	01/02/2016	23/02/2016	17-mar.	26-mar.	04-abr.	11-abr.	15-abr.	21-abr.
DURACION(días)	21	23,45	9,45	9,45	7,7	4,55	6,65	28,7

Tabla 2: Diagrama de Gantt

A.3 ANÁLISIS ECONÓMICO

- **Autor:** Alberto Gutiérrez Pozuelo
- **Departamento de Ingeniería Telemática**
- **Descripción del Proyecto**
  - **Título**: Despliegue de una infraestructura de red definida por software
  - **Duración**: 4,15 meses (jornada 4h/día; 20 h/semanales)
  - **Tasa de costes indirectos:** 20%
- **Presupuesto total del Proyecto:** 10.081,97 €
- **Subcontratación de tareas:** No se especifica
- **Otros costes indirectos:** No se especifica

Concepto	Cantidad	Coste (€)	% Proyecto	Uso (meses)	Depreciación (meses)	Coste imputable(€)
Recursos materiales						
Ordenador portátil	1	620	10	3.94	60	40.71
Trabajadores						
Graduado Ing. Telemática	1 (0.5 ing/mes)	2.694,39 €/mes	-	-	-	1.347,195 €/mes
Ingeniero	1 (0.25 ing/mes)	4.289,54 €/mes	-	-	-	1.072,385 €/mes
SUBTOTAL						**10.041,257 €**
					TOTAL	**10.081,97 €**

Tabla 3: Presupuesto del proyecto

B. MARCO REGULADOR

En la actualidad, y dado el poco nivel de expansión de las redes definidas por software no existe un marco regulador que se pueda aplicar a estas tecnologías. No obstante, se están realizando esfuerzos por anticiparse a su expansión y estar preparados para cuando sea necesario tener disponible un marco regulatorio que aplicar.

En referencia a lo anteriormente comentado, cabe destacar que ya en Enero de 2016 tuvo lugar en Bruselas una conferencia [24] organizada por el Cuerpo de Reguladores Europeos de las Comunicaciones Electrónicas (BEREC, de sus silgas en Inglés "Body of European Regulators of Electronic Communications") dónde algunas de las empresas más influyentes (Open Networking Foundation, Telefonica, Nokia o HP entre otros) expusieron su punto de vista sobre el impacto de las SDN y NFV poniendo el foco en sus implicaciones regulatorias.

En líneas generales, se concluía que aún era temprano para hablar de regulación en las redes definidas por software y que se deben focalizar los esfuerzos en fomentar el desarrollo de estándares abiertos.

BIBLIOGRAFÍA

[1] OpenFlow: http://archive.openflow.org/
[2] Open Networking Foundation(ONF) : https://www.opennetworking.org/about/onf-overview
[3] Virtual Box: https://www.virtualbox.org/
[4] OpenDaylight: https://www.opendaylight.org/
[5] Linux Foundation: http://www.linuxfoundation.org/

[6] Underdahl, Brian & Kinghorn, Gary. "**Software Defined Networking for Dummies, Cisco Special Edition".** Hoboken, New Jersey. John Wiley & Sons, Inc. (2015) Disponible en: http://www.cisco.com/c/dam/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/sdnfordummies.pdf

[7] Open Networking Foundation. "Software-Defined Networking (SDN) Definition". (2016). Disponible en: https://www.opennetworking.org/sdn-resources/sdn-definition

[8] Brent Salisbury. "The Control Plane, Data Plane and Forwarding Plane in Networks" (Septiembre 2012). Disponible en: http://networkstatic.net/the-control-plane-data-plane-and-forwarding-plane-in-networks/

[9] Nadeau, Thomas D. & Gray, Ken. **"SDN: Software Defined Networks"**. Beijing, Cambridge, Farnham, Köln, Sebastopol, Tokyo. O'Reilly Media. (2013).

[10] Diego Kreutz, Fernando M. V. Ramos, Paulo Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, & Steve Uhlig**. "Software-defined networking: A comprehensive survey."**Actas del IEEE (Octubre 2014).

[11] G.Trotter . RFC3222 : "Terminology for Forwarding Information Base (FIB) based Router Performance" . IETF (Diciembre 2001)

[12] Nick Feamster. "Opportunities for Separation of Data and Control" . Curso SDN en Coursera: Módulo 2.2 . Instituto Tecnológico de Georgia. (2015)

[13] Open Networking Foundation. "SDN Architecture Overview". Version1.0. (2013). Disponible en: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/SDN-architecture-overview-1.0.pdf

[14] M. Tim Jones. "La anatomía de un hipervisor Linux". IBM. (2009) Disponible en: http://www.ibm.com/developerworks/ssa/library/l-hypervisor/index.html

[15] Open Networking Foundation. "OpenFlow". (2016). Disponible en: https://www.opennetworking.org/sdn-resources/openflow

[16] Open Networking Foundation. "Member Listing". (2016). Disponible en: https://www.opennetworking.org/our-members

[17] McKeown K., Anderson T., Balakrishnan, H., Parulkar G., Peterson L., Jennifer Rexford, Shenker S. y Turner J. (Marzo, 2008) *OpenFlow: Enabling Innovation in Campus Networks*. Disponible en: http://archive.openflow.org/documents/openflow-wp-latest.pdf

[18] Open Networking Foundation. "OpenFlow Switch Specification". Versión 1.3.3 (versión del protocolo 0x04). ONF TS-015 (Septiembre 2013). Disponible en: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.3.pdf

[19] SDx Central. "What are SDN Controllers (or SDN Controllers Platforms)?". (2016). Disponible en: https://www.sdxcentral.com/sdn/definitions/sdn-controllers/

[20] SDx Central. "What is Network Virtualization? - Definition". (2016). Disponible en: https://www.sdxcentral.com/sdn/resources/sdn-controllers/openflow-controller/

[21] Alejandro García Centeno, Carlos Manuel Rodríguez Vergel, Caridad Anías Calderón, Frank Camilo Casmartiño Bondarenko. "Controladores SDN, elementos para su selección y evaluación". Revista Telemática Vol.13 No.3.p. 10-20 (Septiembre-Diciembre 2014).

[22] Open Virtual Switch Descarga: http://openvswitch.org/download/

[23] OpenDaylight Descargas: https://www.opendaylight.org/downloads

[24] Berec Events 2016. "Public BEREC expert workshop on Regulatory implications of SDN and NFV" Presentaciones disponibles en: http://berec.europa.eu/eng/events/berec_events_2016/104-public-berec-expert-workshop-on-regulatory-implications-of-sdn-and-nfv